ABC INFORMATIQUE

N° 56

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

Commander des robots

Graphiques logo

Les logiciels intégrés

La tablette Koala Pad

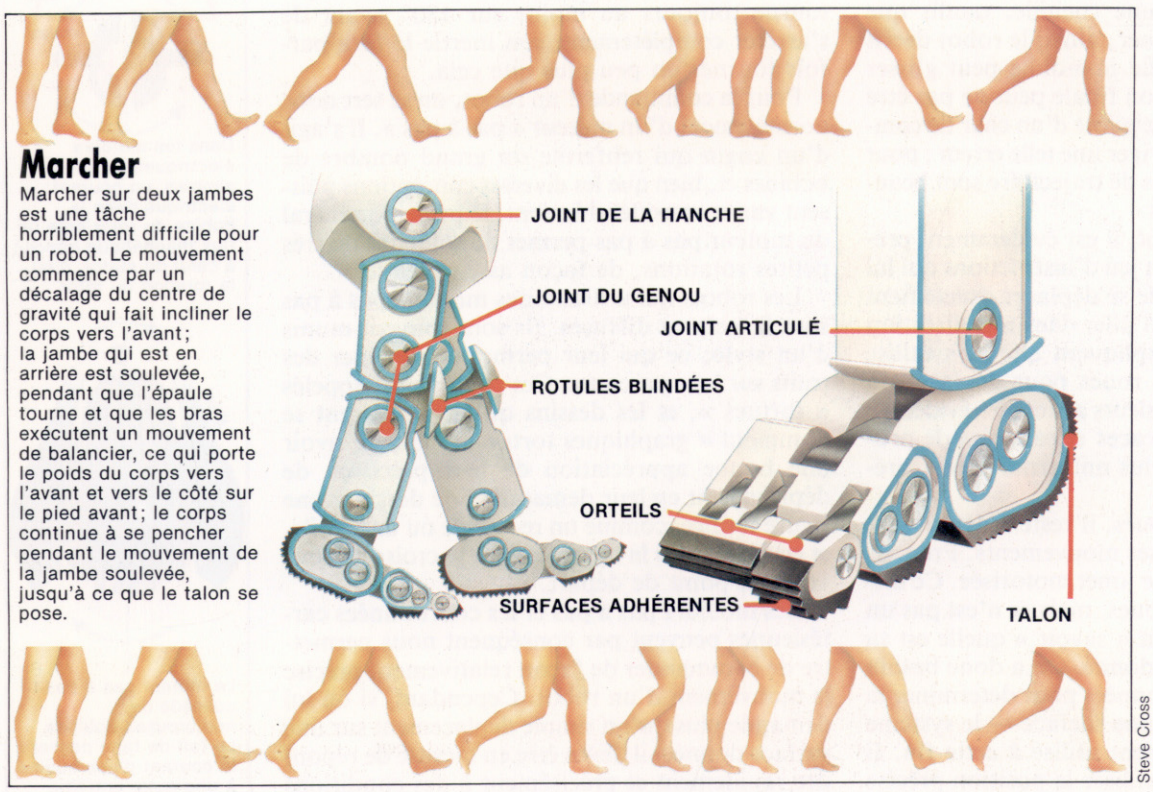
EDITIONS ATLAS

M 6062-56-12-F 85 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

Lève-toi et marche!

Des robots remplacent des ouvriers sur des chaînes de montage. Étudions les trois principales méthodes de déplacement des robots et les moyens les plus efficaces pour les commander.



Steve Cross

Marcher

Marcher sur deux jambes est une tâche horriblement difficile pour un robot. Le mouvement commence par un décalage du centre de gravité qui fait incliner le corps vers l'avant; la jambe qui est en arrière est soulevée, pendant que l'épaule tourne et que les bras exécutent un mouvement de balancier, ce qui porte le poids du corps vers l'avant et vers le côté sur le pied avant; le corps continue à se pencher pendant le mouvement de la jambe soulevée, jusqu'à ce que le talon se pose.

Bien avant qu'un enfant soit en mesure de marcher, il est capable d'aller chercher et de prendre des objets grâce à des déplacements parfois surprenants. Il fait également la démonstration de son intelligence à différentes occasions. Mais l'action de marcher correspond à une étape demandant un long apprentissage. C'est par une pratique de tous les jours que la marche devient automatique.

Il est possible de faire « marcher » les robots, mais les techniques utilisées sont très différentes de celles employées par les êtres humains. Les robots peuvent avoir des jambes qu'ils déplacent vers l'avant ou vers l'arrière, mais chacune de ces jambes est munie d'un pied comportant des roues. Celles-ci sont dotées d'un mécanisme qui interdit tout mouvement vers l'arrière. Un tel robot suit donc une séquence déterminée d'actions pendant qu'il « marche ». Cette méthode comporte un inconvénient majeur : le robot se déplace uniquement en ligne droite et ses mouvements sont imprécis.

Il serait préférable de faire marcher les robots en leur faisant lever d'abord une jambe, puis l'autre, au lieu de leur imprimer un simple mouvement de balancier. Cette approche pose un problème : le robot doit être capable de maintenir son équilibre sur une jambe pendant que l'autre est levée. Plusieurs solutions ont été envisagées pour maintenir le centre de gravité directement au-dessus de la jambe qui porte le poids. Si un tel système était mis au point, les robots pourraient marcher efficacement, et ainsi monter un escalier pour apporter une tasse de thé à leur propriétaire! Mais, en réalité, si monter un escalier ne pose pas un problème insurmontable, la question est de faire comprendre au robot qu'il a atteint le haut de l'escalier. Il a donc besoin d'un équipement supplémentaire pour détecter la dernière marche.

Il serait également possible de faire monter le robot sur des chenilles. Ce système offrirait l'avantage de permettre son déplacement sur un terrain accidenté. L'armée utilise des robots sur chenilles pour transporter des déchets dangereux; ces machines peuvent traverser des zones raisonnablement accidentées.

Les chenilles sont robustes et faciles à diriger, mais elles comportent deux inconvénients principaux. D'abord, comme la plupart des robots

sont assez petits, les chenilles sont étroites, ce qui rend difficile le passage de gros obstacles. Un char de combat peut franchir presque tous les obstacles avec facilité — mais cela n'est possible qu'en raison du volume important des chars, de leur poids et de leur puissance. Si un char essaie de franchir un obstacle d'une dimension telle que son centre de gravité soit déplacé en dehors de la surface portante, il se renverse. Votre robot rencontrera les mêmes problèmes.

Ensuite, les chenilles ne peuvent pas être commandées de façon précise : la direction est donnée par l'arrêt d'une chenille, tandis que l'autre continue à tourner ; ainsi, le robot décrit un arc. Mais la chenille immobile peut glisser légèrement, et la position finale peut ne pas être celle qui était prévue. Le pilote d'un char de combat peut facilement corriger une telle erreur ; pour un robot, les corrections de trajectoire sont beaucoup plus compliquées.

Pour diriger un robot, il est évidemment préférable de disposer d'un jeu d'instructions qui lui permettront toujours de se déplacer exactement vers le bon endroit, et d'aller dans une direction précise. Ces raisons expliquent que l'on utilise plus fréquemment des roues pour déplacer un robot. Elles offrent plusieurs avantages évidents : elles sont simples, efficaces et capables de produire un mouvement plus uniforme et plus précis que des jambes.

Le robot muni de roues, il reste à déterminer comment commander ses mouvements. Prenons par exemple une voiture jouet motorisée. Ce dispositif se déplace sur roues, mais ce n'est pas un robot, puisqu'il ne peut « savoir » quelle est sa position à un moment donné. On a donc besoin d'un système de coordonnées pour déterminer la position de l'objet sur une surface — le système cartésien est généralement utilisé à cette fin. Il est alors possible de localiser la position précise du robot et de spécifier les mouvements nécessaires pour le déplacer. Nous n'avons plus besoin que d'un dispositif qui se charge de faire avancer le robot avec précision dans ce cadre de référence.

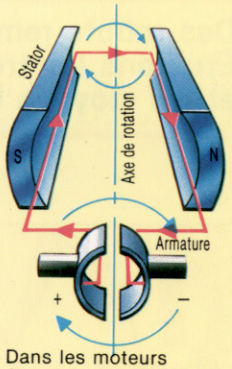
Bien que la puissance hydraulique ou pneumatique soit occasionnellement utilisée, le moteur électrique est généralement retenu pour déplacer les robots. Comme nous l'avons déjà vu, un simple moteur électrique permet de déplacer un objet tout en assurant une modeste commande de sa direction. Cela ne permet pas d'obtenir une maîtrise précise — un moteur électrique simple tourne toujours au moins sur 180° avant de s'arrêter complètement, son inertie le fera parfois tourner un peu plus que cela.

Pour la commande d'un robot, on se sert donc normalement d'un moteur « pas à pas ». Il s'agit d'un engin qui renferme un grand nombre de bobines et, bien que les diverses conceptions puissent varier considérablement, le principe général du moteur pas à pas permet de solliciter de très petites rotations, de façon assez précise.

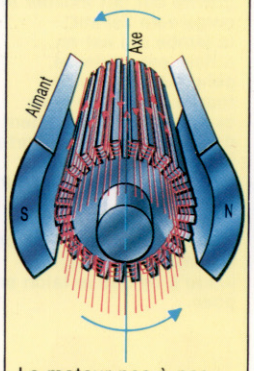
Les robots qui utilisent des moteurs pas à pas sont largement diffusés. Ils sont souvent munis d'un stylo, ce qui leur permet de dessiner des traits sur leur parcours. Ces robots sont appelés « tortues », et les dessins qu'ils produisent se nomment « graphiques tortue ». On peut avoir une bonne appréciation de leur précision de déplacement en leur demandant de dessiner une figure fermée, comme un rectangle ou une étoile, et en vérifiant si la ligne dessinée se croise de nouveau au point de départ.

Les moteurs pas à pas et les coordonnées cartésiennes peuvent par conséquent nous permettre de commander de façon relativement précise le mouvement d'un robot. Cependant, si on lui demande plus qu'un simple déplacement sur une surface donnée, il devra être en mesure de répondre rapidement et précisément à des conditions externes. Nous verrons cela prochainement.

Mesure pas à pas



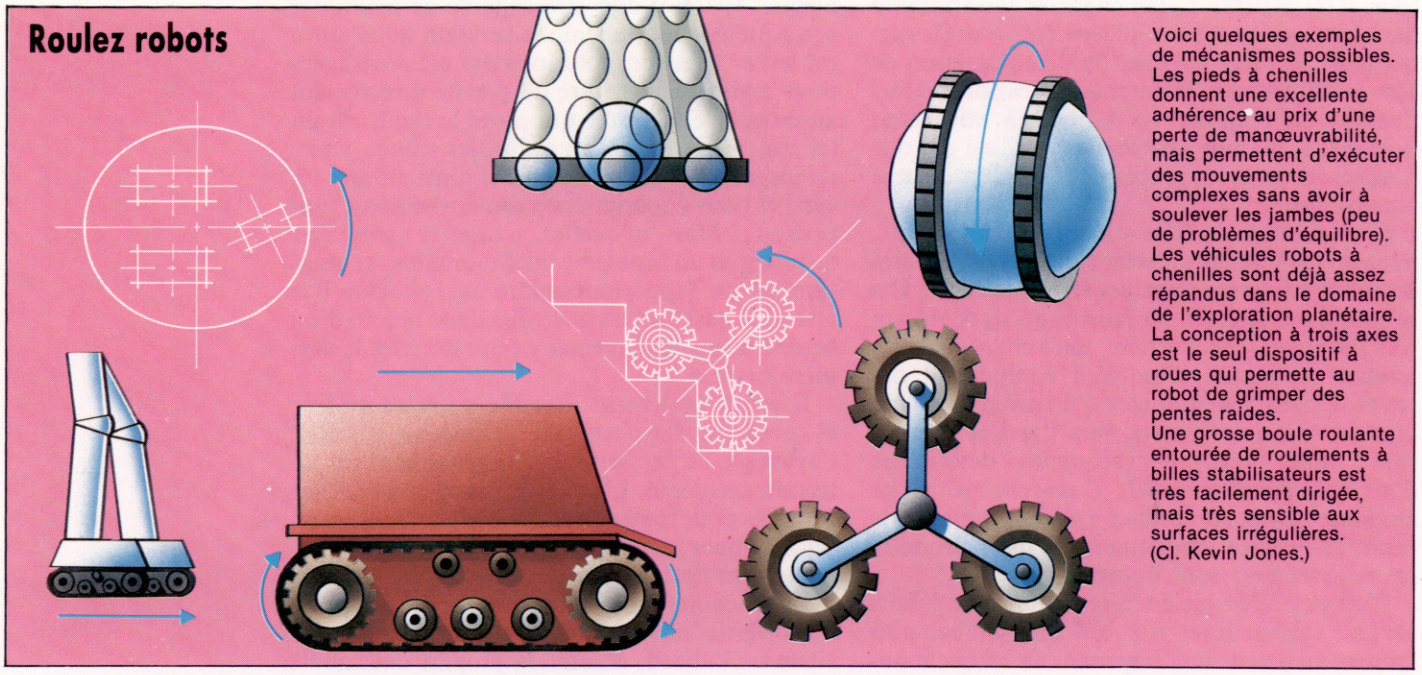
Dans les moteurs électriques les plus simples, un courant électrique dans la bobine du rotor crée un flux magnétique opposé à celui du stator, d'où la rotation du rotor.



Le moteur pas à pas possède de nombreuses bobines. Le fait de faire passer le courant d'une bobine à une autre entraîne une rotation.

Kevin Jones

Roulez robots



Voici quelques exemples de mécanismes possibles. Les pieds à chenilles donnent une excellente adhérence au prix d'une perte de manœuvrabilité, mais permettent d'exécuter des mouvements complexes sans avoir à soulever les jambes (peu de problèmes d'équilibre). Les véhicules robots à chenilles sont déjà assez répandus dans le domaine de l'exploration planétaire. La conception à trois axes est le seul dispositif à roues qui permette au robot de grimper des pentes raides.
Une grosse boule roulante entourée de roulements à billes stabilisateurs est très facilement dirigée, mais très sensible aux surfaces irrégulières. (Cl. Kevin Jones.)

Diviser pour mieux régner

Dans ce cours de LOGO, nous verrons un certain nombre de procédures graphiques destinées à la tortue et qui produisent des effets visuels intéressants au moyen de la récursion.

Notre premier programme est conçu pour dessiner des formes d'arbres. Pour commencer, nous pouvons simplement tracer un tronc avec une branche à droite et une branche à gauche. Les autres branches peuvent ensuite être dessinées de la même manière, avec une tige centrale et des rameaux symétriques. En poursuivant cette ramification nous arrivons à un arbre. C'est un bon exemple de l'utilisation de la récursion avec LOGO.

Notre procédure suppose deux entrées de données : une pour la longueur du tronc et une pour le niveau de ramification. La longueur des branches est diminuée de moitié à chaque ramification.

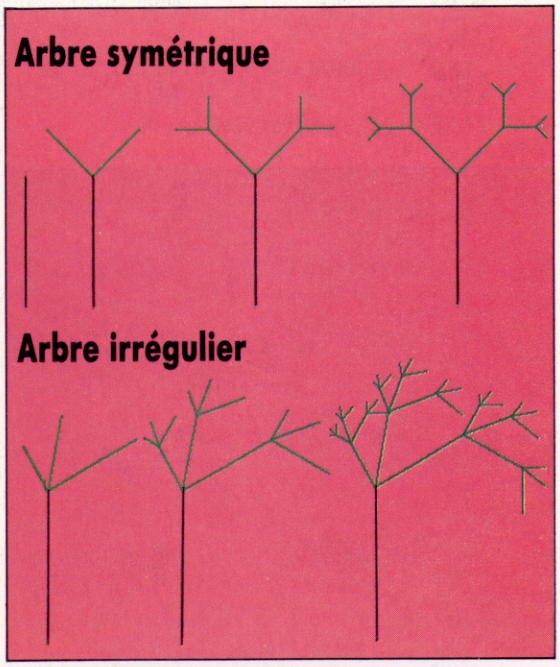
```
POUR BRANCHE :LONGUEUR :NIVEAU
    SI :NIVEAU = 0 ALORS STOP
    AV :LONGUEUR
    GA 45
    BRANCHE ( :LONGUEUR / 2) ( :NIVEAU – 1)
    DR 90
    BRANCHE ( :LONGUEUR / 2) ( :NIVEAU – 1)
    GA 45
    RE :LONGUEUR
FIN
```

Vous remarquez que la procédure est « transparente à l'état » de la tortue, c'est-à-dire que celle-là n'est pas influencée par les divers appels de procédure. Si ce n'était pas le cas, le dessin deviendrait impossible.

Il faut bien reconnaître que cette procédure ne crée pas un arbre très « nature ». Il y a plusieurs façons de modifier la procédure pour qu'elle soit plus intéressante. En voici une qui dessine trois branches, chacune de longueur différente, à chaque niveau de ramification :

```
POUR BRANCHE1 :LONGUEUR :NIVEAU
    SI :NIVEAU = 0 ALORS STOP
    AV :LONGUEUR
    GH 30
    BRANCHE1 ( :LONGUEUR / 3) ( :NIVEAU – 1)
    DR 40
    BRANCHE1 ( :LONGUEUR / 2) ( :NIVEAU – 1)
    DR 50
    BRANCHE1 ( :LONGUEUR / 1,5) ( :NIVEAU – 1)
    GA 60
    RE :LONGUEUR
FIN
```

Essayez d'autres modifications pour dessiner des arbres plus « vrais ».



Polygones à damier

Cette procédure trace un carré, le divise en quatre, puis chaque partie en quatre...

```
POUR DAMIER :LONGUEUR :NIVEAU
    SI :NIVEAU = 0 ALORS RÉPÈTE 4 [AV :LONGUEUR DR 90]
      STOP
    DAMIER ( :LONGUEUR / 2) ( :NIVEAU – 1)
    AV ( :LONGUEUR / 2)
    DAMIER ( :LONGUEUR / 2) ( :NIVEAU – 1)
    DR 90
    AV ( :LONGUEUR / 2)
    GA 90
    DAMIER ( :LONGUEUR / 2) ( :NIVEAU – 1)
    RE ( :LONGUEUR / 2)
    DAMIER ( :LONGUEUR / 2) ( :NIVEAU – 1)
    GA 90
    AV ( :LONGUEUR / 2)
    DR 90
FIN
```

Écrivez une procédure similaire qui éclate un triangle en quatre plus petits, puis divise chacun à nouveau en quatre, et ainsi de suite.

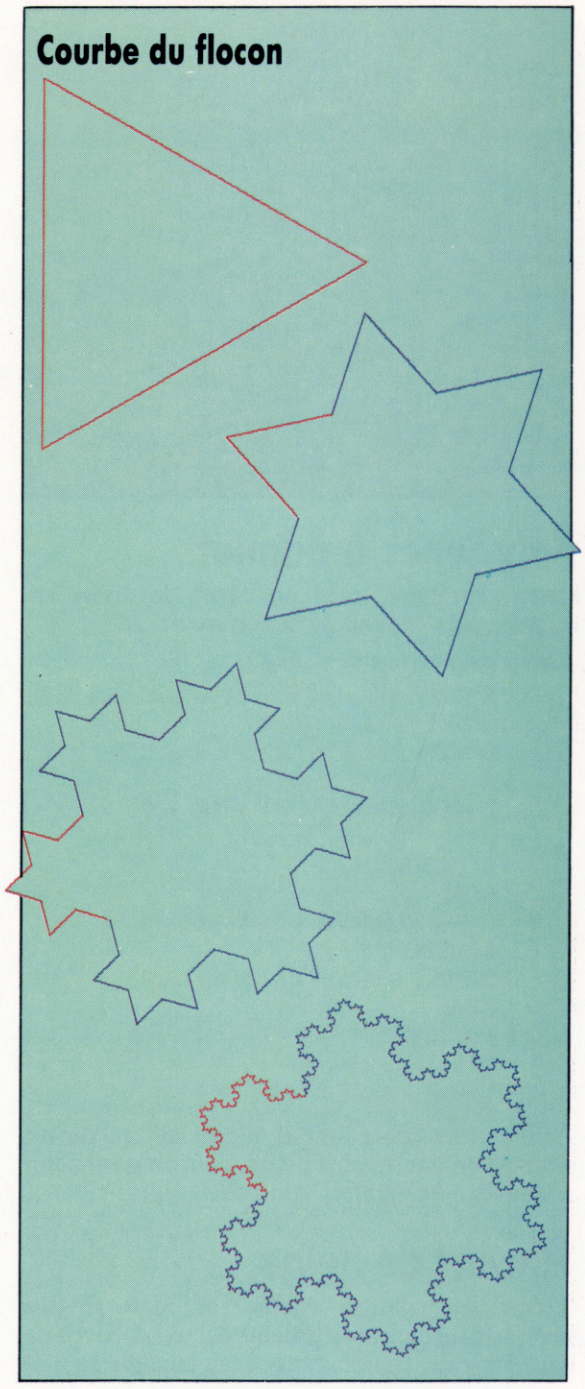
Flocons de neige

Tracez un triangle équilatéral, puis divisez chaque côté en trois segments égaux. A partir de ces segments, tracez un nouveau triangle équila-

téral vers l'extérieur. Effacez le côté commun avec le grand triangle. Vous obtenez une étoile à six branches. Répétez l'opération, et vous obtenez le « flocon » :

```
POUR FLOCON :LONGUEUR :NIVEAU
      RÉPÈTE 3 [CÔTÉ :LONGUEUR :NIVEAU DR 120]
FIN

POUR CÔTÉ :LONGUEUR :NIVEAU
      SI :NIVEAU = 0 ALORS AV :LONGUEUR STOP
      CÔTÉ ( :LONGUEUR / 3) ( :NIVEAU – 1)
      GA 60
      CÔTÉ ( :LONGUEUR / 3) ( :NIVEAU – 1)
      DR 120
      CÔTÉ ( :LONGUEUR / 3) ( :NIVEAU – 1)
      GA 60
      CÔTÉ ( :LONGUEUR / 3) ( :NIVEAU – 1)
FIN
```



Vous remarquez que la procédure CÔTÉ n'est pas « transparente », c'est-à-dire que la tortue est influencée par elle. Mais cette procédure a été écrite pour laisser la tortue dans la position appropriée pour tracer le côté suivant.

Si la division des côtés du triangle se poursuit indéfiniment, le résultat obtenu est une courbe de longueur infinie qui pourtant entoure une surface définie ! Il est possible de prouver que cette courbe n'est ni unidimensionnelle ni bidimensionnelle, mais quelque chose entre les deux !

Une courbe similaire peut être obtenue à partir d'un carré, en divisant chaque côté en trois parties égales, et ainsi de suite. Écrivez la procédure pour ces carrés.

La suite de courbes montrée ici a été inventée par le mathématicien Sierpinski. Lorsque l'on poursuit l'opération indéfiniment, on obtient une courbe (une ligne unidimensionnelle), qui traverse chaque point du carré externe (une forme bidimensionnelle). Il existe d'autres courbes qui remplissent ainsi l'espace.

La procédure utilisée est assez complexe. La courbe de premier niveau est constituée de quatre côtés (en bleu), réunis par quatre diagonales (en rouge). La procédure principale, appelée SIERP, divise simplement en quatre parties le traitement des côtés, de sorte que la procédure UN.CÔTÉ ne traite qu'un côté à la fois.

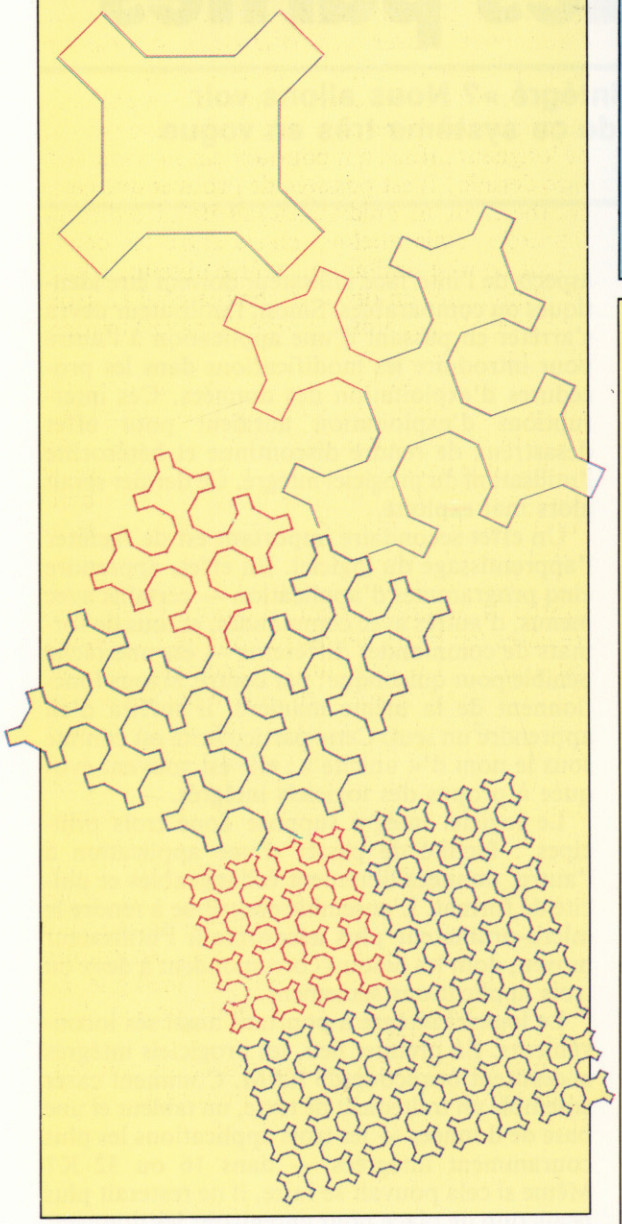
Étudions la question pour un seul côté : il est constitué de trois lignes, une diagonale, une horizontale ou verticale, et une autre diagonale. Au niveau 2, chaque diagonale est remplacée par une autre diagonale plus petite, constituée d'un ensemble de trois lignes, et la ligne horizontale ou verticale est remplacée par deux ensembles similaires de trois lignes réunis par une ligne. Le même processus est répété pour passer d'un niveau à un autre.

Voici les procédures pour tracer les courbes. Vous remarquez comment la commande FAIRE est utilisée pour initialiser DIAG :

```
POUR SIERP :CÔTÉ :NIVEAU
      FAIRE «DIAG :CÔTÉ / DRCR (côté droit du carré)
      RÉPÈTE 4[UN.CÔTÉ:NIVEAU DR 45 AV :DIAG DR 45]
FIN

POUR UN.CÔTÉ :NIVEAU
      SI :NIVEAU = 0 STOP
UN.CÔTÉ ( :NIVEAU – 1)
      DR 45
      AV :DIAG
      DR 45
      UN.CÔTÉ ( :NIVEAU – 1)
      GA 90
      AV :CÔTÉ
      GA 90
      AV :CÔTÉ
      GA 90
      UN.CÔTÉ ( :NIVEAU – 1)
      DR 45
      AV :DIAG
      DR 45
      UN.CÔTÉ ( :NIVEAU – 1)
FIN
```

Courbe de Sierpinski



Variantes de LOGO

Les versions LOGO LCSI utilisent DÉTPOS (DÉTerminer POSition) pour positionner la tortue. Cela suppose une liste en entrée et les deux coordonnées doivent être combinées à la commande LIST. Par exemple :

```
DÉTPOS LIST 45 67
```

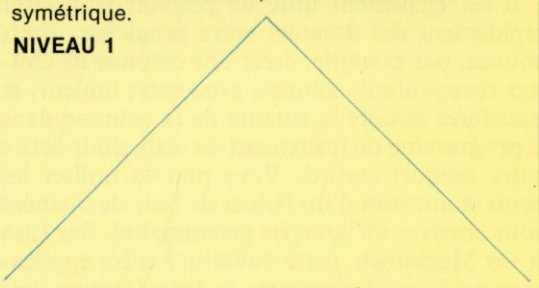
Souvenez-vous également que les versions LCSI utilisent une syntaxe différente avec SI. Une règle d'arrêt (STOP) sera par exemple :

```
SI NIVEAU = 0 [STOP]
```

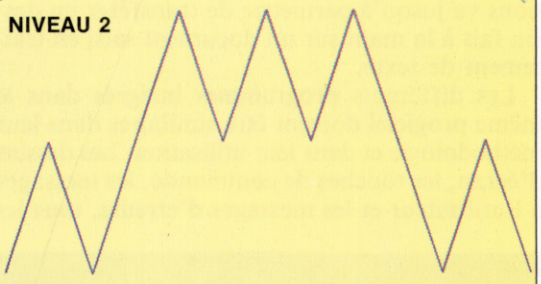
Problème de procédure n° 4

Notre illustration représente diverses formes géométriques qui définissent des courbes à l'infini. Le premier niveau comprend une ligne qui monte et une qui descend. Pour passer au niveau suivant, la ligne ascendante est remplacée par six lignes égales et en dents de scie : le premier tronçon monte à mi-hauteur, le deuxième redescend complètement; le troisième et le quatrième s'enchaînent pour atteindre la hauteur complète du sommet précédent, et le dernier redescend à mi-hauteur. Les angles entre les montées et les descentes sont toujours les mêmes. La ligne descendante suit la même démarche, et l'ensemble est symétrique.

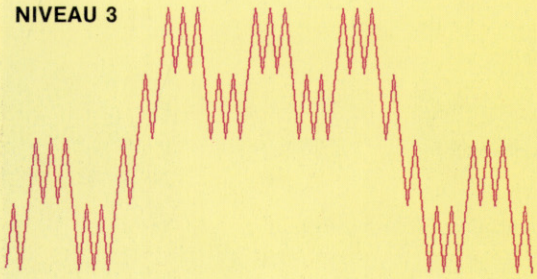
NIVEAU 1



NIVEAU 2



NIVEAU 3



Écrivez les procédures pour cette série de courbes. Utilisez POSXY au lieu de AV et DR. La procédure principale commencera par diviser l'opération en deux parties : une pour la ligne montante, et une pour la ligne descendante. Vous écrirez ensuite une procédure pour chacune de ces deux parties. Souvenez-vous que ces procédures peuvent s'appeler mutuellement et s'appeler elles-mêmes !

Réponses aux exercices

Procédure récursive traçant une pile de carrés :

```
POUR TOUR :HAUTEUR
        SI :HAUTEUR < 5 ALORS STOP
        CARRÉ :HAUTEUR
        DÉPLACEMENT :HAUTEUR
        TOUR ( :HAUTEUR / 2)
FIN

POUR CARRÉ :LONGUEUR
        RÉPÈTE 4 [AV :LONGUEUR DR 90]
FIN

POUR DÉPLACEMENT :LONGUEUR
        AV :LONGUEUR
        DR 90
        AV ( :LONGUEUR / 4)
        GA 90
FIN
```

La somme des parties

Que signifie l'expression « logiciel intégré »? Nous allons voir les avantages et les inconvénients de ce système très en vogue dans le monde du logiciel.

L'intégration représente une des tendances les plus passionnantes de l'évolution des logiciels. Si elle concerne essentiellement les micro-ordinateurs de gestion, elle commence à parvenir aux micros personnels.

La plus grande réussite de l'intégration est de permettre au programmeur de commuter directement et rapidement entre plusieurs progiciels. Un système idéal devrait même permettre de ne pas avoir à quitter un programme, à revenir au système d'exploitation et à permuter les disques, pour en commencer un autre. Pour être efficace, le changement d'application doit se faire sur simple frappe d'une touche.

Il est également utile de pouvoir transférer rapidement des données entre progiciels. Vous pouvez, par exemple, créer une colonne de chiffres récapitulatifs annuels avec votre tableur, et transférer ensuite la totalité de la colonne dans le programme de traitement de texte pour écrire votre rapport annuel. Vous pouvez utiliser les noms et adresses d'un fichier de base de données pour envoyer un courrier personnalisé. Sur Lisa et sur Macintosh, cette ouverture entre applications va jusqu'à permettre de transférer un dessin fait à la main sur un document saisi en traitement de texte.

Les différents programmes intégrés dans le même progiciel doivent être similaires dans leur méthodologie et dans leur utilisation. Les dessins d'écran, les touches de commande, les messages à l'utilisateur et les messages d'erreurs, tous les aspects de l'interface utilisateur doivent être identiques ou comparables. Sinon, l'utilisateur devra s'arrêter en passant d'une application à l'autre pour introduire les modifications dans les procédures d'exploitation des données. Ces interruptions d'exploitation auraient pour effet désastreux de rendre discontinue et hétéroclite l'utilisation du progiciel intégré. Ce dernier serait alors mal exploité.

Un effet secondaire important est de faciliter l'apprentissage du logiciel. En effet, apprendre cinq programmes d'application — certains avec menus, d'autres avec commandes, et tous de formats de commandes différents — est une tâche pénible pour quiconque; par contre, si tous fonctionnent de la même manière, il suffira d'en apprendre un seul. Cette particularité est connue sous le nom d'« unicité »; elle est souvent évoquée à propos des logiciels intégrés.

Le logiciel intégré suppose donc trois principes : facilité de passer d'une application à l'autre, données librement échangeables et unicité de format. L'ensemble contribue à rendre le micro-ordinateur plus accessible à l'utilisateur moyen, dont les besoins correspondent à deux ou trois applications seulement.

Le logiciel intégré a pourtant aussi ses inconvénients. En premier lieu, les progiciels intégrés nécessitent beaucoup de RAM. Comment caser ensemble un traitement de texte, un tableur et une base de données — les trois applications les plus couramment intégrées — dans 16 ou 32 K? Même si cela pouvait se faire, il ne resterait plus beaucoup de place pour enregistrer les données. Ce handicap limite les logiciels intégrés aux machines de gestion, de grande capacité mémoire (128 K en général). Bien sûr, les programmes intégrés peuvent partager certaines routines, et les opérations de sauvegarde ou de gestion interne ne seront écrites qu'une seule fois. Mais il n'en reste pas moins que les applications en elles-mêmes ont des besoins spécifiques et incompressibles qui prennent de la place en RAM.

Un autre point faible des logiciels intégrés provient également de ce problème de place mémoire. Pour gagner sur celle-ci, les auteurs de logiciels intégrés ont eu recours à des simplifications sur chaque programme. Ainsi, un traitement de texte intégré avec d'autres programmes d'application ne pourra être aussi complet qu'un traitement de texte seul. Ce dernier peut disposer de beaucoup de mémoire, pratiquement autant à lui tout seul que tous les programmes d'application réunis.

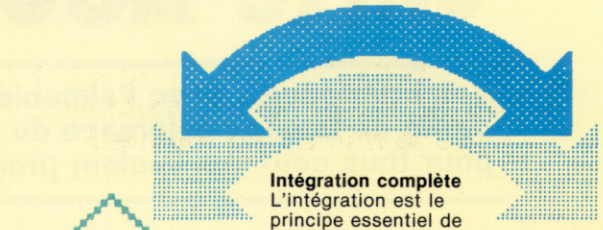
A passer en revue... Parmi les progiciels intégrés destinés d'abord aux professionnels, mais qui peuvent intéresser un plus large public, le Lotus 1-2-3. (Cl. Ian McKinnell/ Microscope.)

Prenons comme exemple un programme fonctionnant sur l'IBM PC et ses compatibles : Multimate est conçu sur la base du traitement de texte de Wang. Il comporte de nombreuses options de formatage et de saisie de documents que l'on ne retrouve pas dans des programmes plus petits. Avec lui, la création de documents complexes de grande taille est assez simple. Multimate nécessite en lui-même 192 K de RAM. Lotus, un logiciel intégré comprenant traitement de texte, tableur et base de données, demande également 192 K. Mais la place mémoire utilisée avec Multimate pour une seule application correspond, avec Lotus, à la place nécessaire à trois applications complètes.

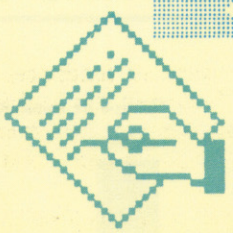
Un troisième inconvénient de l'intégration logicielle résulte... de l'un de ses points forts. Nous avons vu qu'il est important pour les programmes intégrés de se ressembler pour être facilement assimilables et exploitables. Malheureusement, les auteurs de logiciels doivent, là aussi, trouver des compromis. Un tableur ne se manie pas comme une base de données ou comme un traitement de texte, chacun ayant ses propres modes opérationnels. Aussi, les éléments nécessaires à l'aboutissement de chaque programme doivent cohabiter en un ensemble le plus continu possible. Microsoft a rencontré ce problème pour la conception de Microsoft Word, logiciel de traitement de texte uniquement. En effet, il fallait que le dessin d'écran et le déroulement du programme soient compatibles avec le tableur Multiplan. Le but était de pouvoir intégrer facilement les deux programmes, dont Multiplan était le fer de lance. Microsoft prévoyait pour Word un menu affiché en permanence, à la manière de Multiplan. Plus tard, Microsoft s'aperçut que les utilisateurs, qui trouvaient ce genre de menu très utile pour un tableur, le trouvaient carrément encombrant pour un traitement de texte.

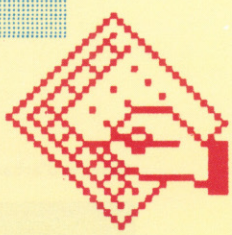
La règle d'or à propos d'un logiciel est d'abord de plaire et de satisfaire aux besoins de l'utilisateur. Si une même personne a plusieurs tâches à concilier, comme écrire une lettre, faire des calculs simples ou tenir un petit fichier, un progiciel intégré peut grandement simplifier les choses. Toutefois, il faut parfois faire des sacrifices. Quelqu'un qui, par exemple, désirerait écrire un roman ou des textes de reportage assez longs à l'aide d'un micro-ordinateur ne devrait pas trop espérer dans les logiciels intégrés pour différentes tâches. Il aura tout intérêt à faire appel à des programmes séparés de traitement de textes, tableurs ou base de données. Et remarquons que les auteurs de logiciels écrivent des instructions de plus en plus ramassées et synthétiques, sur une place mémoire de plus en plus restreinte; dans la mesure où la mémoire elle-même tend à s'accroître sur les micro-ordinateurs personnels, le logiciel intégré est appelé à se développer pour tous les micros, et pas seulement pour les micro-ordinateurs de gestion. Des micros grand public le proposent déjà. Nous verrons prochainement, plus en détail, certains programmes intégrés qui ont marqué et marquent encore le développement des programmes.

Règles du jeu



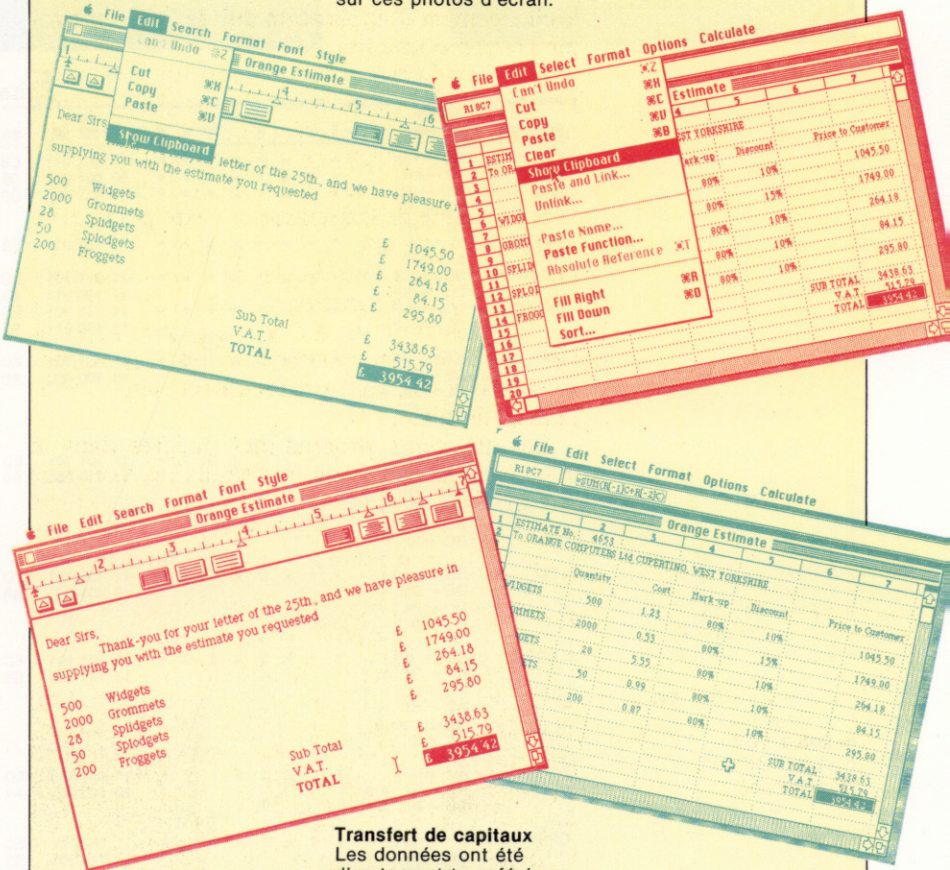
Intégration complète
L'intégration est le principe essentiel de l'exploitation du Macintosh d'Apple, d'où proviennent ces illustrations (en anglais). Multiplan (le tableur), MacWrite (le traitement de texte) et MacPaint (le programme graphique) communiquent directement entre eux par l'intermédiaire du système d'exploitation.





MacWrite Multiplan

Unicité du code
L'unicité de format entre les applications intégrées apparaît nettement ici, sur ces photos d'écran.



Transfert de capitaux
Les données ont été directement transférées depuis le tableur sur le traitement de texte.

Ian McKinnell

Nous nous arrêterons en particulier sur une approche de l'intégration, que l'on rencontre sur des machines telles que Lisa et Macintosh, qui reconsidèrent tout l'environnement du système d'exploitation, en fonction de l'intégration. Mais nous n'oublierons pas qu'il existe d'autres produits très intéressants sur le marché!

Carte Z80 (suite)

Voici reproduite, avec l'aimable autorisation de Zilog Inc., une autre partie de la carte référence du programmeur Z80. A ne pas manquer pour tous ceux qui veulent programmer.

Groupe d'échange

		Adressage à progression automatique avancée				
		AF′	BC′, DE′ & HL′	HL	IX	IY
Prog. autom. avancée	AF	08				
	BC, DE & HL		D9			
	DE			EB		
Registre indirect	(SP)			E3	DD E3	FD E3

Groupe de transfert de bloc

HL pointe la source
DE pointe la destination
BC est le compteur d'octets

DESTINATION		SOURCE: REG. INDIR. (HL)	
Reg. indir.	(DE)	ED A0	'LDI'—Load (DE) ← (HL) Inc HL & DE, Dec BC
		ED B0	'LDIR'—Load (DE) ← (HL) Inc HL & DE, Dec BC, Repeat until BC = 0
		ED A8	'LDD'—Load (DE) ← (HL) Dec HL & DE, Dec BC
		ED B8	'LDDR'—Load (DE) ← (HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Groupe de recherche de bloc

HL pointe l'emplacement en mémoire à comparer au contenu de l'accumulateur
BC est le compteur d'octets

EMPLACEMENT DE RECHERCHE: REG INDIR. (HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR'—Inc HL, Dec BC Repeat until BC = 0 or find match
ED A9	'CPD'—Dec HL & BC
ED B9	'CPDR'— Dec HL & BC Repeat until BC = 0 or find match

Échange, transfert de bloc et recherche

Mnémonique	Opération symbolique	Drapeaux S	Z	H	P/V	N	C	Opc 76 543 210	Hex	Nombre d'octets hex	Nombre de cycles M	Nombre d'états T	Commentaires
EX DE HL	DE → HL	•	•	×	•	×	•	•	•	11 101 011	EB	1	1	4	Échange de banque de registre et banque de registre auxiliaire
EX AF AF	AF → AR	•	•	×	•	×	•	•	•	00 001 000	08	1	1	4	
EXX	BC → BC DE → DE HL → HL	•	•	×	•	×	•	•	•	11 011 001	D9	1	1	4	
EX (SP) HL	H → (SP + 1) L → (SP)	•	•	×	•	×	•	•	•	11 100 011	E3	1	5	19	
EX (SP) IX	IX_H → (SP + 1) IX_L → (SP)	•	•	×	•	×	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP) IY	IY_H → (SP + 1) IY_L → (SP)	•	•	×	•	×	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1	•	•	×	0	×	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Charge (HL) dans (DE), incrémente les pointeurs et décrémente le compteur d'octets (BC)
LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1 Repeat until BC = 0	•	•	×	0	×	0	0	•	11 101 101 11 110 000	ED B0	2 2	5 4	21 16	SI BC ≠ 0 SI BC = 0
LDD	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1	•	•	×	0	×	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1 Repeat until BC = 0	•	•	×	0	×	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	Si BC ≠ 0 Si BC = 0 Si BC = 0 ou A = (HL)
CPI	A ← (HL) HL ← HL + 1 BC ← BC − 1	↕	② ↕	×	↕	×	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A ← (HL) HL ← HL + 1 BC ← BC − 1 Repeat until A = (HL) ou BC = 0	↕	② ↕	×	↕	×	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	
CPD	A ← (HL) HL ← HL − 1 BC ← BC − 1	↕	② ↕	×	↕	×	① ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A ← (HL) HL ← HL − 1 BC ← BC − 1 Repeat until A = (HL) ou BC = 0	↕	② ↕	×	↕	×	① ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	Si BC ≠ 0 et A ≠ (HL) Si BC = 0 ou A = (HL)

Notes : 1. Le drapeau P/V est nul si le résultat de BC — 1 = 0, sinon P/V = 1
2. Le drapeau Z est 1 si A = (HL), sinon Z = 0

Notation de drapeau : • = drapeau non affecté ; 0 = drapeau à zéro ; 1 = drapeau mis ;
X = drapeau inconnu ; ↕ = drapeau affecté selon le résultat de l'opération.

Du bout des doigts

Si vous êtes possesseur d'un Commodore 64, la tablette graphique Koala Pad vous aidera à surmonter l'absence totale d'instructions relatives au graphisme.

Le Commodore 64 a d'excellentes possibilités graphiques. Pourtant, il reste encore dépourvu d'aides à la création. Sans doute est-ce lié à la manipulation, assez difficile, du mode haute résolution de l'appareil. La compagnie américaine Koala Technologies a mis au point une tablette graphique qui vous facilitera la tâche.

Contrairement à bien des périphériques du même genre, le Koala Pad est léger, compact, et ne mesure que 20,5 cm sur 16 cm. On remarque au centre un carré de 11 cm de côté, fait de fibres de carbone, et en dessous duquel se trouve une membrane sensible à la moindre pression. L'utilisateur peut, à l'aide d'un stylet en plastique, d'un crayon, ou d'un doigt, commander le déplacement d'un curseur sur l'écran.

La membrane est en fait composée de fils conducteurs, disposés en deux couches : l'une pour l'axe horizontal, l'autre pour l'axe vertical. A chaque pression, l'appareil repère les fils qui entrent en contact, et obtient ainsi des coordonnées qu'il transmet à l'ordinateur. Deux boutons sont installés au sommet de la tablette; il est indispensable d'appuyer sur l'un d'eux avant de colorier un point sur l'écran, ou de choisir l'une des nombreuses options disponibles. On peut se servir, indifféremment, de l'un ou de l'autre : les gauchers ne seront donc pas pénalisés.

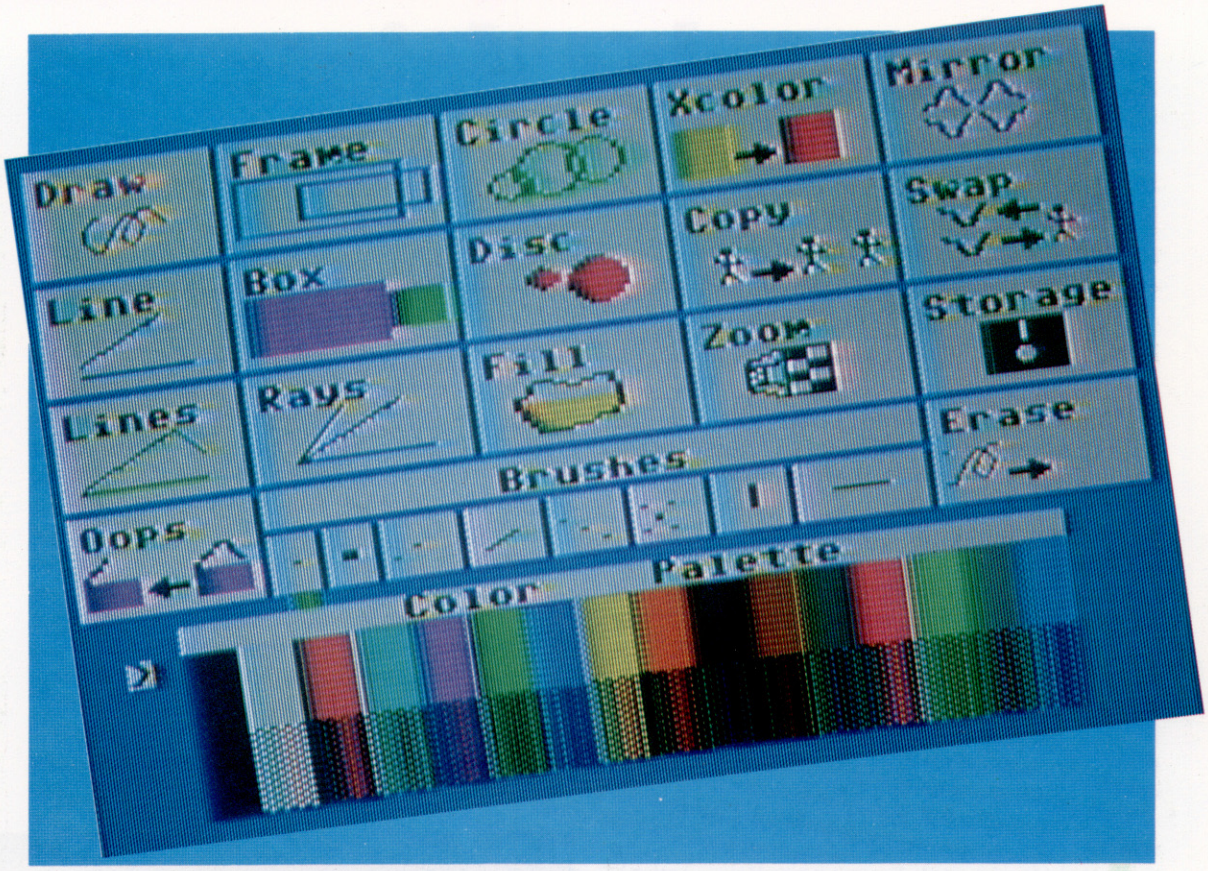
Mise en couleur

La tablette est connectée au Commodore 64 par l'intermédiaire du port manches à balai; il faut ensuite charger le logiciel Koala Painter à partir d'un lecteur de disquettes. Cette opération menée à bien, l'écran affiche l'ensemble des commandes. Vous pourrez vous servir d'une palette comportant les seize couleurs habituelles, plus seize « trames », formées de pixels de couleurs différentes, ce qui donne des effets de dégradés très intéressants. Juste au-dessus de la palette, huit cases différentes abritent chacune un « pinceau » de taille variable. Ce sont, en fait, des formes qui peuvent être tracées à l'écran, et qui vont du simple pixel à des combinaisons de points et de lignes. Viennent ensuite les options dont vous pouvez faire usage. Pour en choisir une, il suffit d'appuyer sur la membrane, ce qui permet de guider le déplacement du curseur (une flèche). Lorsqu'il est juste au-dessus de la commande qui vous intéresse, pressez l'un des boutons et elle sera aussitôt activée (elle se met alors à clignoter, de façon à vous éviter tout risque d'erreur ou d'oubli). Koala Painter permet la création de lignes, de rayons (lignes tracées à partir du même point de départ), de cercles et de formes géométriques simples, tandis que BOX fait naître des carrés de couleur, et DISC des cercles également coloriés. Une zone donnée — il faut bien sûr qu'elle soit close — peut être mise en couleur grâce à FILL. X-COLOR entraîne un changement de telle ou telle teinte.

Avec la commande MIRROR, l'écran est divisé en quatre parties; le curseur ne peut plus se déplacer que dans le quadrant supérieur gauche, mais tout ce qu'il trace est aussi reproduit dans les trois autres sections, de façon à former un motif symétrique.

La commande la plus puissante reste cependant ZOOM. L'utilisateur peut sélectionner n'importe

A portée de main
On peut, en quelques heures, mettre au point des graphismes très complexes, grâce à un logiciel guidé par menu et très facilement compréhensible. A la différence de bien des tablettes graphiques, le Koala Pad peut être tenu en main pendant qu'on en fait usage.
(Cl. Ian McKinnell.)



Faites votre choix
Le menu principal est constitué de cases, chacune correspondant à une option, dont le nom (ici en anglais) est affiché, accompagné d'une « icône » explicative. De ce point de vue, certaines illustrations sont peut-être un peu sibyllines. Il faut déplacer la flèche-curseur au-dessus d'une case, puis appuyer sur un des boutons. Le nom de la commande se met alors à clignoter, afin que l'utilisateur ne perde pas de vue le mode qu'il a choisi.
(Cl. Ian McKinnell.)

quelle partie de l'écran, et la voir agrandie dans une « fenêtre » placée en bas. Chaque pixel est grossi huit fois, et a donc la taille d'un caractère standard. Il est alors possible de faire des corrections très minutieuses. Tout ce que vous dessinez peut aussi être reproduit n'importe où sur l'écran, grâce à la commande COPY, et en plusieurs exemplaires s'il le faut.

Lorsque vous aurez choisi une option, appuyez sur un des deux boutons pour accéder enfin à votre « toile » (en fait, l'écran graphique du Commodore 64), et mettez-vous aussitôt à la tâche en jouant du curseur et du bouton de sélection.

Quelques limitations

Le maniement de la tablette est enfantin. La maîtrise des commandes est vite acquise. Les résultats sont excellents, parfois comparables à ceux de bien des logiciels du commerce. Malheureusement, la commande DRAW (qui vous permet de dessiner à votre gré) se révèle souvent insuffisamment précise. En effet, la membrane a une résolution inférieure à celle du Commodore 64; et plus d'une fois le stylet (ou l'ongle) appuiera en fait sur deux points à la fois. L'ordinateur, en essayant d'analyser ces données, tracera bien un point, mais pas toujours à l'endroit désiré. Autre détail gênant : pour changer de couleur, il faut obligatoirement retourner au menu principal, sauf si l'on est alors en train d'utiliser la commande ZOOM. Ce sont là, il faut le préciser, de simples critiques de détail; à l'inverse, les commandes FILL et LINE sont exécutées à une allure impressionnante.

En revanche, les créateurs du Koala Pad auraient pu trouver un moyen plus habile de corriger les erreurs, qui sont annulées par la commande OOPS; on y accède à partir du menu principal. Mais elle est trop puissante, et supprime tout ce qui a été créé depuis que l'on est sorti pour la dernière fois de ce même menu. A la limite, une seule erreur suffit à anéantir le travail d'une heure entière. Bien entendu, il est toujours possible de faire appel à ZOOM et d'effectuer la correction pixel par pixel, mais cela prend du temps, surtout s'il faut effacer ou déplacer un bloc ou un disque de couleur mal placés. Il serait sans doute judicieux de restreindre la portée de OOPS, qui n'aurait plus d'effet qu'à partir de la dernière pression sur un des boutons de la tablette graphique.

Raccordé au port manches à balai, le Koala Pad peut donc remplacer celui-ci, et l'utilisateur aura donc tout loisir de s'en servir au sein de ses propres programmes. Pour connaître la position du curseur à partir du BASIC, il faudra lire le contenu des adresses mémoire 54297 et 54298, grâce à une simple instruction PEEK, de façon à obtenir, respectivement, les coordonnées de X et de Y.

Vous pourrez sauvegarder sur disquette les images créées, et les intégrer dans vos programmes, ce qui devrait permettre la création de jeux d'aventures graphiques avec un texte en bas de l'écran et une illustration au-dessus. Le Koala Pad et son logiciel permettent la sauvegarde et le rappel d'un maximum de seize écrans différents, de 8 K chacun. Il n'est pas possible d'en charger un directement en mémoire; mais le

manuel comporte un petit programme BASIC qui rend la chose possible.

Un dessin quelconque créé avec Koala Painter et sauvegardé ensuite, ne peut avoir qu'une résolution maximale de 255 × 255 pixels (il est sans doute possible de faire un peu mieux en langage machine). Il y a là un problème, puisque le Commodore 64 a, lui, une résolution maximale de 320 × 200 pixels (il est vrai, en deux couleurs seulement). Le lecteur aura sans doute vu tout de suite que 255 a été retenu, parce que c'est le plus grand nombre adressable à l'aide d'un seul octet : un écran complet exigerait un adressage à 16 bits. Il arrive malheureusement qu'une partie du dessin soit perdue : tant qu'il n'est pas affiché directement, il est conservé en mémoire, au sein de la RAM utilisateur. De plus, il doit être transféré, avec toutes les informations relatives à sa couleur, au lieu de son stockage dans la mémoire d'écran haute résolution, qui commence à l'adresse 55296.

Ajoutons encore qu'il est regrettable que le logiciel d'accompagnement ne soit disponible que sur disquette, ce qui limite le marché de cette tablette, pourtant bien utile.

Toutefois, on peut dire que Koala Pad est un périphérique très utile pour quiconque voudrait produire des graphiques haute résolution sur le Commodore 64, en particulier pour les artistes ou les « aventuriers » de la nouvelle écriture.

KOALA PAD

PRIX

★ ★ ★

DIMENSIONS

210 × 165 mm.

ÉCRAN

L'écran haute résolution du Commodore 64 (320 × 200 pixels) est mis en œuvre, bien qu'il soit limité à 255 × 255 pixels, quand l'image est appelée par un programme BASIC.

INTERFACE

Se branche sur l'ordinateur au port manches à balai.

DOCUMENTATION

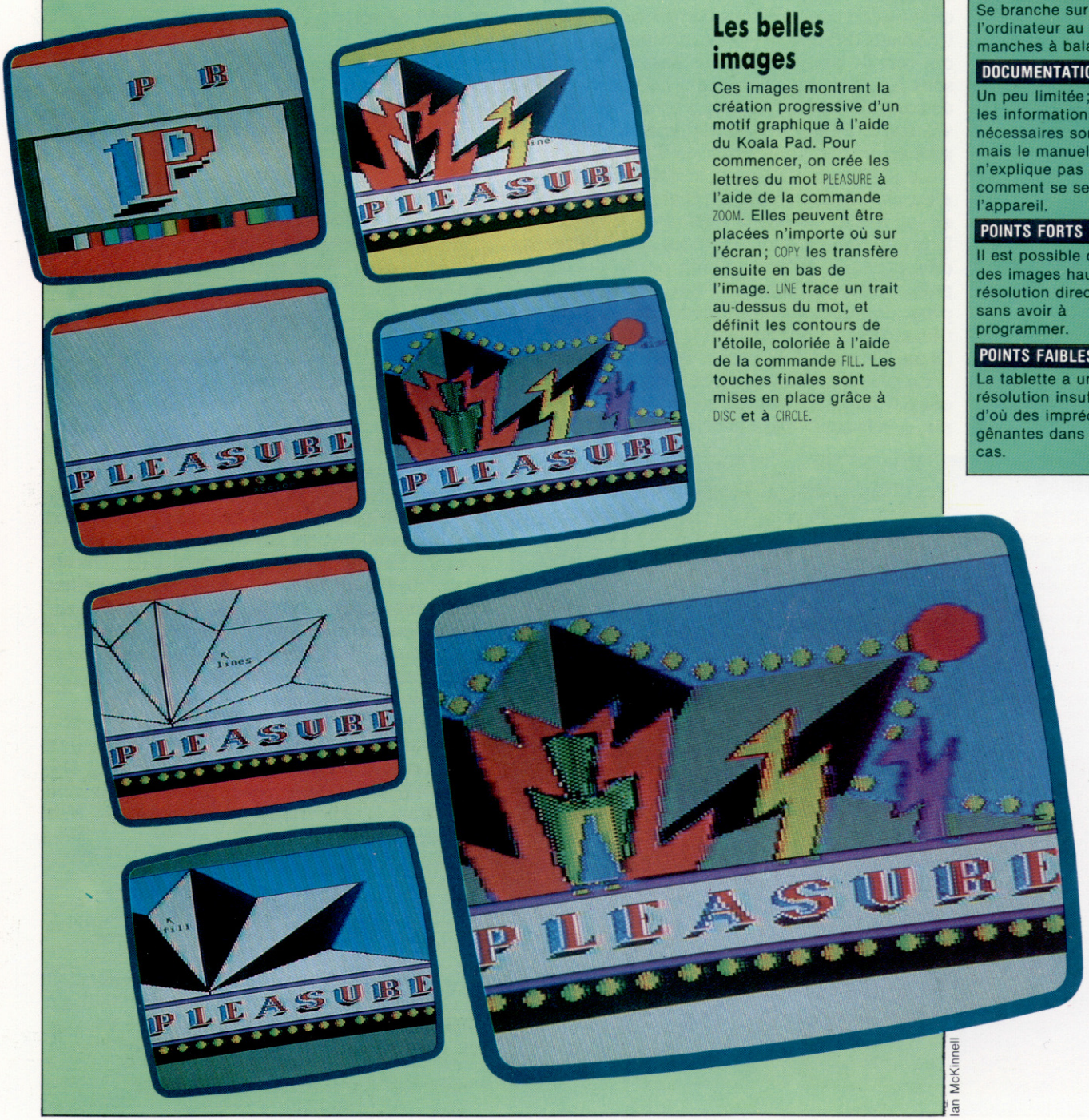
Un peu limitée; toutes les informations nécessaires sont là, mais le manuel n'explique pas vraiment comment se servir de l'appareil.

POINTS FORTS

Il est possible de créer des images haute résolution directement, sans avoir à programmer.

POINTS FAIBLES

La tablette a une résolution insuffisante, d'où des imprécisions gênantes dans certains cas.



Les belles images

Ces images montrent la création progressive d'un motif graphique à l'aide du Koala Pad. Pour commencer, on crée les lettres du mot PLEASURE à l'aide de la commande ZOOM. Elles peuvent être placées n'importe où sur l'écran; COPY les transfère ensuite en bas de l'image. LINE trace un trait au-dessus du mot, et définit les contours de l'étoile, coloriée à l'aide de la commande FILL. Les touches finales sont mises en place grâce à DISC et à CIRCLE.

Ian McKinnell

Sur votre vélo

Le jeu qui suit est assez simple et très amusant : il ne prend que trente-cinq lignes BASIC. Il s'agit d'un jeu à deux joueurs, alors n'hésitez pas.

L'action de ce jeu, qui rappelle une scène de *Tron,* un film de Walt Disney, se déroule en champ clos; chaque joueur dispose d'une sorte de « vélo fou » qui se déplace à une vitesse incroyable et ne peut être arrêté. Le seul contrôle qui vous est concédé vous permet de prendre des tournants de 90° à la vitesse maximale. Les vélos ont ceci de particulier qu'ils laissent une trace, sous forme de mur lumineux. Le but du jeu est de contraindre l'adversaire à venir s'écraser sur vos traces à l'intérieur du labyrinthe de plus en plus étroit que vous tissez sur l'écran.

Le jeu est ici réalisé sur le ZX Spectrum, qui n'est pas particulièrement réputé pour la vitesse de son BASIC. Comme il s'agit d'un jeu d'action, le programme a d'autres visées que l'élégance; cela explique qu'une grande partie du listage puisse sembler décousue. Les appels de sous-programmes et autres procédés de programmation structurée ont été évités, dans la mesure où ils auraient nui à la vitesse d'exécution.

La première étape consiste à créer l'aire de jeu et à définir l'affichage du score. C'est assez simple, et cela contribue à la brièveté du programme final. Le seul point digne d'être remarqué est le bord de la zone, qui se situe à un caractère en dedans par rapport à la zone d'écran disponible. Cette précaution est nécessaire, afin que le graphisme d'une collision avec le mur de bordure de zone ne sorte pas de l'écran :

```
10 LET p=0: LET q=0
100 BORDER 0: PAPER 0: CLS
110 PRINT AT 0,1; INK 6; « Vélo un= »;q
120 PRINT AT 0,19; INK 5; « Vélo deux= »; p
130 INK 2
140 PLOT 8,8: DRAW 239,0: DRAW 0,159
150 DRAW -239,0: DRAW 0,-159
```

L'arène est en rouge, le jaune est pour le premier vélo et le cyan, pour le second. Les variables p et q contiennent respectivement le score des deux vélos. L'étape suivante consiste à initialiser toutes les variables.

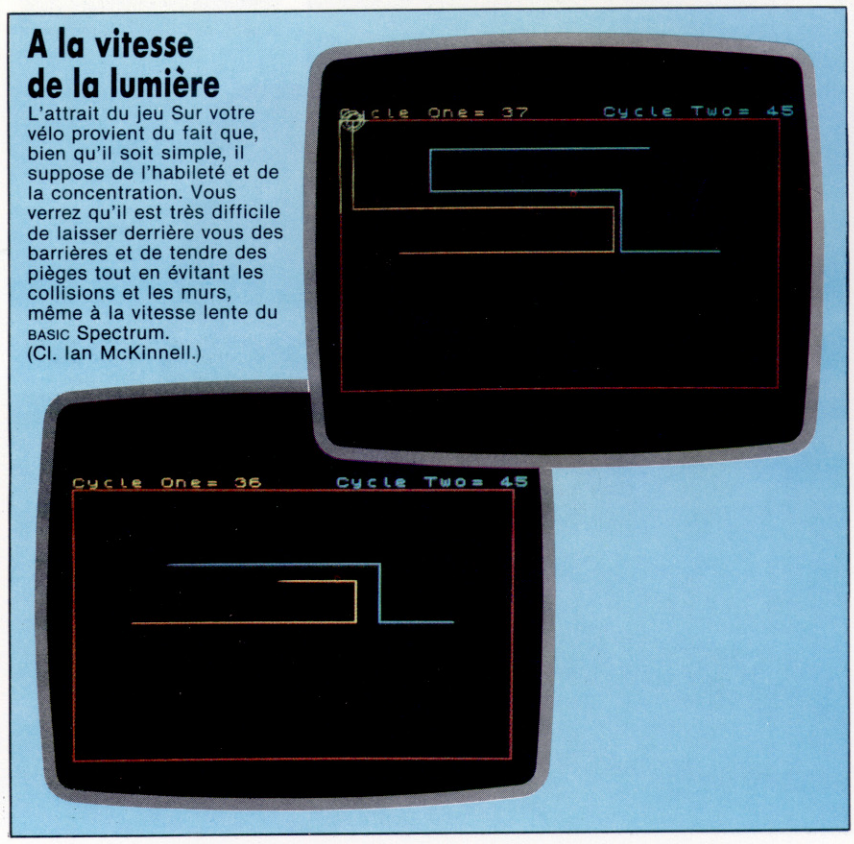
Il nous faut maintenant commencer à nous faire une idée de la mise en œuvre de l'action principale du jeu. Pour un seul vélo, c'est assez simple — comme cela est montré dans l'organigramme. POINT nous permet de savoir si la position courante du vélo est déjà prise (par un mur ou par l'autre vélo) et, si oui, de nous brancher sur la routine de collision. Dans le cas contraire, la position est gagnée par l'intermédiaire de PLOT (tracer). Le clavier est ensuite lu afin de détecter un éventuel changement de direction. La position est ensuite augmentée d'une unité dans la direction courante, et le cycle reprend. Il nous faut donc quatre variables, deux pour les coordonnées courantes X et Y, et deux pour la direction courante d'après l'axe des X et des Y.

Mais nous devons gérer deux vélos se déplaçant simultanément. Une solution élégante consisterait à utiliser deux tableaux à deux éléments x(2) et y(2) pour les positions, mais cela ralentirait le jeu. Aussi utiliserons-nous huit variables distinctes :

```
200 LET x=40: LET y=88
210 LET m=215: LET n=88
220 LET a=1: LET b=0
230 LET i=-1: LET j=0
```

Cela initialise les vélos (détermine leur position de départ) et les met en route l'un vers l'autre à raison de 1 pixel à la fois. L'action de base du jeu est donc assez simple à développer :

```
400 IF POINT (x,y)=1 THEN LET col=6: GOTO 700
410 IF POINT (m,n)=1 THEN LET col=5:
```



A la vitesse de la lumière
L'attrait du jeu Sur votre vélo provient du fait que, bien qu'il soit simple, il suppose de l'habileté et de la concentration. Vous verrez qu'il est très difficile de laisser derrière vous des barrières et de tendre des pièges tout en évitant les collisions et les murs, même à la vitesse lente du BASIC Spectrum.
(Cl. Ian McKinnell.)

```
    LET x=m: LET y=n: GOTO 700
420 PLOT INK 6;x,y:PLOT INK 5;m,n
```

(Les lignes 500 à 570 constituent la routine qui détermine les nouvelles valeurs de a, b, i et j.)

```
600 LET x=x+a: LET y=y+b
610 LET m=m+i: LET n=n+j
620 GOTO 400
```

Le seul passage peu clair est peut-être la ligne 410, où les variables pour le vélo n° 1 sont attribuées également au vélo n° 2 et où une nouvelle variable, col, est introduite. L'explication vient du fait qu'une seule routine peut servir pour la collision, avec x et y indiquant le point de collision, et col la couleur.

La routine de contrôle de la saisie au clavier devrait être rapide, mais il nous faut malheureusement utiliser IF...THEN, qui est assez lent. Nous pouvons cependant utiliser la commande rapide IN pour lire les touches. Les touches de contrôle choisies sont Q et A pour contrôler le déplacement du vélo n° 1 vers le haut et vers le bas; P et ENTER pour le vélo n° 2. La gauche et la droite sont X et C pour le vélo n° 1; N et M pour le vélo n 2.

```
500 IF IN 64510=190 THEN LET a=0: LET b=1
510 IF IN 65022=190 THEN LET a=0: LET b=-1
520 IF IN 65278=187 THEN LET a=-1: LET b=0
530 IF IN 65278=183 THEN LET a=1: LET b=0
540 IF IN 57342=190 THEN LET i=0: LET j=1
550 IF IN 49150=190 THEN LET i=0: LET j=-1
560 IF IN 32766=187 THEN LET i=1: LET j=0
570 IF IN 32766=183 THEN LET i=-1: LET j=0
```

Il ne reste à écrire que la routine de collision et la mise à jour des scores. Une suite de cercles concentriques, centrés sur le point d'impact, sont choisis avec des rayons de 4, 6 et 8 pixels.

```
700 FOR d=1 TO 3
710 CIRCLE BRIGHT 1; INK col; x,y,2+d*2
720 NEXT d
730 IF col=6 THEN LET p=p+1: GOTO 750
740 LET q=q+1
750 GOTO 100
```

Cela met fin au jeu, la dernière instruction bouclant sur les procédures d'initialisation du début. Le jeu serait néanmoins davantage accessible avec une procédure de début :

```
300 PRINT AT 10,5; INK 7; « APPUYEZ SUR UNE TOUCHE POUR
    COMMENCER »
310 IF INKEY$=«» THEN GOTO 310
320 PRINT AT 10,5;«
```

On obtient ainsi un temps de latence entre deux parties. Il ne reste plus qu'à sauvegarder le jeu sur cassette, de préférence avec SAVE « Sur votre vélo » LINE 10, de sorte que le jeu s'exécute automatiquement dès qu'il est chargé.

Le jeu pourrait évidemment être plus passionnant. On pourrait aussi choisir l'option de jouer avec un seul joueur. Enfin, de meilleurs graphiques et un meilleur son seraient les bienvenus. Mais tout cela rendrait le jeu très lent et peu jouable.

Applaudissements à une seule main

Mise en œuvre
Dans la version « un seul joueur », l'ordinateur est le joueur n° 1. La partie du programme qui lit les mouvements au clavier pour le joueur n° 1 est évitée. L'algorithme du code transmis alors permet au joueur n° 1 de poursuivre en ligne droite jusqu'à ce qu'il provoque un déplacement interdit ou jusqu'à ce que le deuxième joueur change de direction. Dans ce dernier cas, la direction prise par le joueur n° 1 devient l'image en miroir de celle du joueur n° 2. La nouvelle direction est vérifiée. Lorsqu'une direction interdite est détectée pour le joueur n° 1, le programme en cherche une nouvelle, à angle droit sur la gauche ou sur la droite. S'il n'en est pas trouvé, la position du joueur n° 1 est considérée comme fatale. Les modifications que nous apportons ici permettent de choisir entre les versions à deux joueurs et à un seul, au début de chaque nouvelle partie. Il n'est pas difficile d'écrire des algorithmes satisfaisants pour le jeu, mais il est très difficile de les développer en BASIC sans ralentir considérablement l'action. Apportez les modifications suivantes au jeu :

```
20 LET touchebd=500:LET
   pt=-1:LETinchangé=0:
   RANDOMIZE
260 PRINT AT 10,5;«Nbre de joueurs
   (1/2)?»
270 LET a$=INKEY$:IF a$<>«1» AND a$
   <>«2»
   THEN GOTO 270
280 IF INKEY$<>«2»THEN
   LET touchebd=440
```

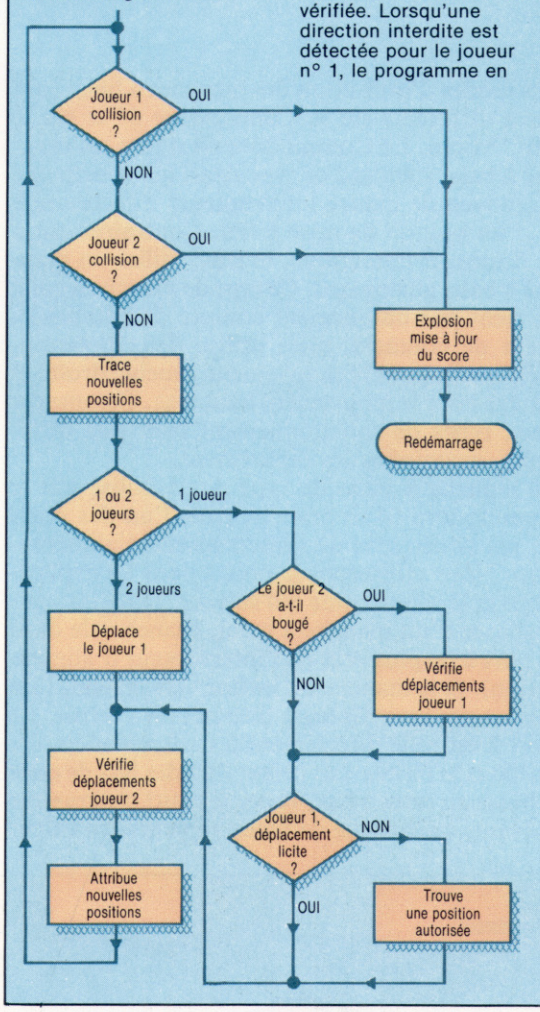
Ces lignes de code donnent à l'utilisateur le choix du type de jeu et développent ce dernier, soit en accédant à la stratégie « un seul joueur » résidant entre les lignes 440 et 560, soit à la version standard « deux joueurs », entre les lignes 500 et 570. Notre stratégie est dans ces lignes :

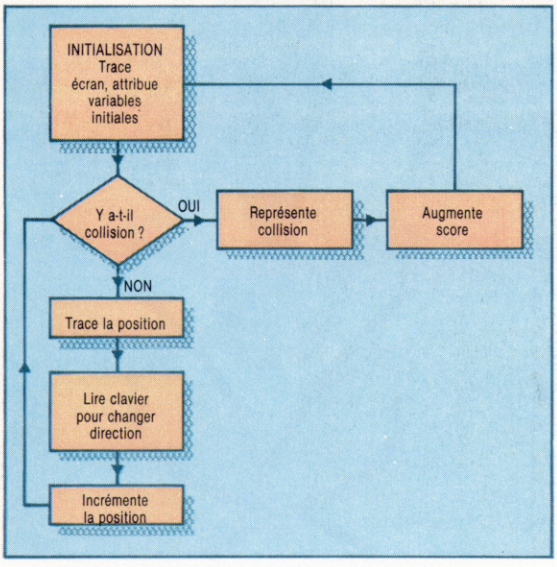
```
430 GOTO touchebd
440 LET pt=SGN(RND-0.5):IF
   inchangé=ptTHEN LET a=pt*j:
   LET b=pt*i:LET inchangé=0
450 IF POINT (x+a,y+b)<>1
   THEN GOTO 540
460 LET pt=SGN (RND-.5):LET
   a=a*pt: LET b=b*pt: LET d=a:
   LET a=b: LET b=d: IF POINT
   (x+a,y+b)<>1 THEN GOTO
   540
490 LET a=-a: LET b=-b: GOTO
   540
540 IF IN 57342=190 THEN LET
i=0: LET j=1: LET inchangé=1
```

et, de même, ajoutez :

```
:LET inchangé=1
```

jusqu'à la fin des lignes 550 à 570.





Version à un seul joueur
Cet organigramme simplifié représente la structure du programme réduite à un seul joueur. Chaque traitement est répété pour la version complète à deux joueurs. (Cl. Liz Dixon.)

Contrôle de trafic

Nous avons construit un système d'entrée/sortie pour commander une voiture LEGO munie de deux moteurs. Nous mettons maintenant ce véhicule sous le contrôle d'un manche à balai.

Les modes d'utilisation des manches à balai sont assez différents sur le Commodore 64 et sur le BBC Micro. Le Commodore 64 utilise un manche à balai standard de type Atari qui fonctionne au moyen de quatre interrupteurs directionnels et d'un bouton de mise à feu. Contrairement à ce dispositif numérique, le BBC utilise un manche à balai analogique. Ce type de manche à balai ne tient pas uniquement compte de l'établissement de contacts, mais utilise deux potentiomètres, un pour le mouvement gauche/droite et l'autre pour le mouvement haut/bas. Comme ces deux types de fonctionnement sont très différents, examinons-les séparément.

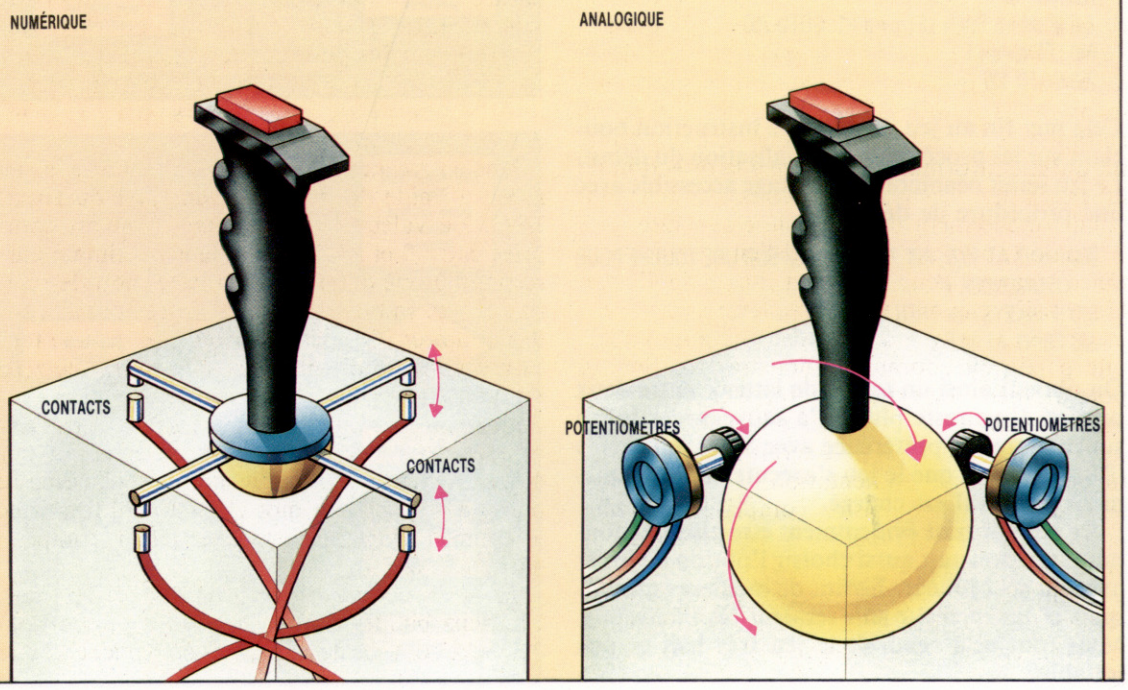
Le manche à balai de type Atari utilisé par le Commodore 64 se branche dans l'un des ports de jeu situés près de l'interrupteur d'alimentation. Nous utiliserons le port 2 dans les explications et le programme qui suivent. Donc, si vous disposez d'un manche à balai, branchez-le dans le port 2 (le plus près de l'interrupteur d'alimentation). Si le manche est incliné vers le haut, l'un des quatre interrupteurs internes sera fermé. Le port 2 est relié directement à l'adresse mémoire 56320 et le fermer fait passer l'un des bits de cette adresse au niveau bas. Tapez le court programme suivant, qui affiche de façon répétitive la valeur du registre de données. Lors de l'exécution du programme, déplacez le manche à balai et appuyez sur le bouton de mise à feu, en notant les divers changements qui surviennent dans la valeur affichée à l'écran.

```
10 REM **** MANCHE A BALAI CBM 64 ****
20 PORT 2=56320
30 MANCHE = PEEK(PORT2) : GOSUB 500
40 PRINT CHR$(145);MANCHE,B$
50 GOTO30
60 :
500 REM SP CONVERSION BINAIRE
510 B$="":N=MANCHE
520 FOR D=1 TO 8
530 N1=INT(N/2):R=N-2*N1
540 B$=B$+STR$(R):N=N1
550 NEXT D
560 RETURN
```

Après quelques minutes d'expérimentation, vous devez pouvoir déterminer quels bits de l'adresse du manche à balai correspondent aux quatre interrupteurs de direction et au bouton de mise à feu. Normalement, lorsque le manche à balai se trouve en position centrale, le contenu de son adresse est 127, c'est-à-dire 01111111. Pousser le manche fait passer cette valeur à 126 (01111110). Le bit 0 est évidemment connecté à l'interrupteur de direction en haut du manche. Nous pouvons résumer l'effet des interrupteurs sur le contenu de l'adresse du manche à balai de la façon suivante :

Analogique et numérique

Le manche à balai numérique ne fait qu'enregistrer le déplacement du manche dans l'une des quatre directions; la valeur du déplacement n'est pas mesurée. Le manche à balai analogique utilise des potentiomètres disposés en quadrature pour mesurer le déplacement du manche, et communique l'information sous forme de deux tensions variables. Ces tensions sont converties en un nombre par le convertisseur analogique numérique.



Kevin Jones

Manche à balai	Décimal	Binaire
Central	127	01111111
Haut	126	01111110
Bas	125	01111101
Gauche	123	01111011
Droite	119	01110111
Feu	111	01101111

Vous avez peut-être également noté que le déplacement du manche à balai en diagonale peut fermer deux interrupteurs simultanément. Bien que nous n'ayons pas besoin d'une détection de mouvement en diagonale pour commander notre véhicule, les résultats de tels mouvements sur l'adresse mémoire seraient les suivants :

Manche à balai	Décimal	Binaire
Haut et gauche	122	01111010
Haut et droite	118	01110110
Bas et gauche	121	01111001
Bas et droite	117	01110101

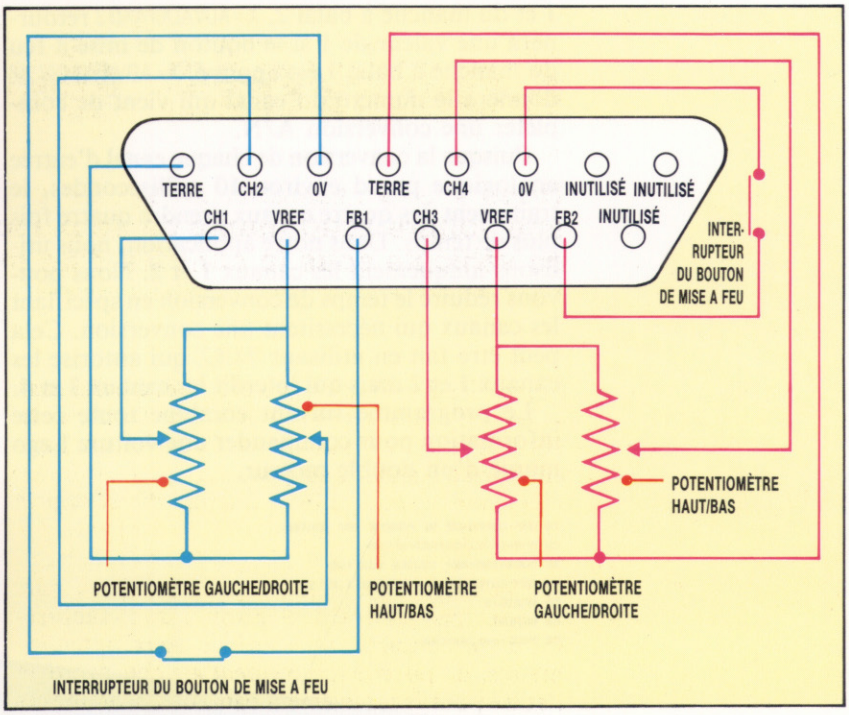
Le programme suivant utilise un manche à balai pour commander les mouvements d'un véhicule à deux moteurs. Le véhicule doit être connecté au boîtier de sortie, et le manche à balai doit être branché au port 2, situé du côté droit du Commodore 64.

```
10 REM MANCHE A BALAI CBM 64
20 DDR=56579:DATREG=56577
30 POKEDDR,255:REM TOUTES SORTIES
40 MANCHE=PEEK(56320):REM MANCHE PORT2
50 GOSUB 1000:REM TEST MANCHE A BALAI
60 POKE DATREG,0:GOTO40
90 :
1000 REM SP TEST MANCHE A BALAI
1005 IF MANCHE=127 THEN POKEDATREG,0
1010 IF MANCHE=126 THEN POKEDATREG,5
1020 IF MANCHE=125 THEN POKEDATREG,10
1030 IF MANCHE=123 THEN POKEDATREG,6
1040 IF MANCHE=119 THEN POKEDATREG,9
1050 IF MANCHE=111 THEN POKEDATREG,0:END
1060 RETURN
```

Le manche à balai du BBC est un dispositif analogique qui obtient l'information relative aux mouvements haut/bas et droite/gauche à l'aide de deux potentiomètres. La différence essentielle entre les deux dispositifs, numérique et analogique, réside dans le fait que ce dernier donne une information relative à une position située entre certaines limites, tandis que le premier ne fournit qu'une information relative à la direction de la trajectoire. Les potentiomètres du manche à balai fonctionnent de la manière suivante.

Un potentiomètre est essentiellement une résistance à travers laquelle une tension est appliquée. Un troisième connecteur au potentiomètre peut être déplacé le long de la résistance pour capter une partie du courant d'alimentation. Sur un potentiomètre de type linéaire, si la connexion mobile était positionnée au milieu de la résistance, la tension résultante serait égale à la moitié de la tension d'alimentation. Il est donc possible d'obtenir une tension comprise entre zéro et la tension d'alimentation en déplaçant la connexion centrale.

Dans un manche à balai analogique, le mouvement de la connexion centrale est provoqué par le déplacement du manche.



Fonctionnement analogique
Les manches à balai du BBC Micro sont généralement fournis par paires, allant dans un seul connecteur.
Le brochage du port analogique, que l'on voit de l'extérieur de la machine, montre que le manche à balai 2 est connecté de façon exactement similaire au manche à balai 1.

Les manches à balai du BBC Micro sont généralement fournis par paires. Les connexions au port analogique pour les manches à balai 1 et 2 sont illustrées dans le schéma ci-dessus.

Une tension de référence est fournie par le micro à chaque potentiomètre et la tension produite par la connexion centrale est introduite par deux canaux d'entrée. Le canal 1 est utilisé pour l'entrée provenant du potentiomètre gauche/droite et le canal 2 sert à accepter l'entrée provenant du potentiomètre haut/bas. Le bouton de mise à feu est un simple interrupteur.

Dès que les entrées du potentiomètre ont été acceptées, les valeurs analogiques sont converties en valeurs numériques par un convertisseur interne. Cette conversion est effectuée en comparant la tension d'entrée avec la tension de référence. Le temps de conversion est d'environ 10 millisecondes pour chaque canal lu. Dès que les informations sont sous forme numérique, nous pouvons utiliser ces valeurs pour commander notre véhicule.

Une entrée au port analogique peut être lue en BASIC à l'aide de la commande ADVAL du BASIC BBC. La valeur retournée par ADVAL est comprise entre 0 et 65520; la valeur maximale correspond à une entrée égale à la tension de référence. Les faibles tensions d'entrée produiront des nombres plus petits, jusqu'à ce qu'une tension d'entrée nulle produise une valeur de zéro retournée par ADVAL. Pour cette simple application, nous ne nous sommes, en fait, intéressés qu'aux deux valeurs limites. Le canal lu par ADVAL est déterminé par le nombre placé entre parenthèses après le mot clé. ADVAL (1) lira donc le canal 1 et retournera une valeur comprise entre 0 et 65520.

ADVAL (0) effectue deux fonctions différentes. Les deux bits les moins significatifs correspondent aux boutons de mise à feu du manche à balai

1 et du manche à balai 2. X=ADVAL(0) AND 3 retournera une valeur de 1 si le bouton de mise à feu du manche à balai 1 est appuyé. X=ADVAL(0) DIV 256 donnera le numéro du canal qui vient de compléter une conversion A/N.

Puisque la conversion de chaque canal d'entrée analogique prend environ 10 millisecondes, le traitement des quatre canaux prendra quatre fois plus de temps. Dans notre application, nous utilisons uniquement les canaux 1 et 2. Nous pouvons réduire le temps de conversion en spécifiant les canaux qui nécessitent une conversion. Cela peut être fait en utilisant *FX16,2, qui autorise les canaux 1 et 2 mais qui interdit les canaux 3 et 4.

Le programme suivant combine toute cette information pour commander une voiture Lego munie d'un double moteur.

```
10 REM COMMANDE DE MANCHE BBC CONTROL
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=255:REM TOUTES SORTIES
40 REM AUTORISE A-N CANAUX 1 ET 2
50 *FX16,2
60 REPEAT
70 PROCtest_manche
80 UNTIL feu=1
90 END
100 :
110 DEF PROCtest_manche
120 REPEAT
130 channel=ADVAL(0, DIV 256
140 UNTIL channel<>0:REM ATTENDRE CONVERSION
150 IF CHANNEL=1 THEN PROCgauche_droite
160 IF CHANNEL=2 THEN PROChaut_bas
170 ENDPROC
180 :
190 DEF PROCgauche_droite
200 REPEAT
210 VALmanche=ADVAL(1)
220 IF valmanche<100 THEN ?DATREG=9
230 IF valmanche>640000 THEN ?DATREG=6
240 feu=ADVAL(0) AND 8
250 PRINT?DATREG,channel,valmanche
260 UNTIL(valmanche<100 AND valmanche<64000 OR feu
270 ?DATREG=0
280 ENDPROC
290 :
300 DEF PROChaut_bas
310 REPEAT
320 valmanche=ADVAL(2)
330 IF valmanche<100 THEN ?DATREG=10
340 IF valmanche 64000 THEN ?DATREG=5
350 feu=ADVAL(0) AND 8
360 PRINT?DATREG,channel,valmanche
370 UNTIL(valmanche>100 AND valmanche<64000)OR feu
380 ?DATREG,0
390 ENDPROC
```

Réponses aux exercices

1) L'étalonnage de votre véhicule peut être réalisé en mesurant le temps nécessaire pour parcourir diverses distances, par exemple 10 cm, 20 cm, 50 cm, 100 cm et 150 cm. En calculant la vitesse sur chaque distance et en faisant une moyenne, il est possible d'obtenir une bonne estimation de la distance parcourue en une seconde. On peut adopter une approche similaire pour les virages, en sélectionnant de nombreux angles et en chronométrant. La commande des moteurs par mise hors et sous tension peut présenter de nombreuses difficultés : la structure du programme de commande est telle que les intervalles de temps doivent être calculés aussi précisément que possible. Des différences de synchronisation de quelques centièmes de seconde peuvent produire de grandes divergences dans les distances parcourues ou dans les angles de virage. Ces problèmes peuvent être réduits considérablement en introduisant un mécanisme de démultiplication entre le moteur et les roues d'entraînement.

2) Retracer le parcours en sens inverse est un peu compliqué. Nous devons d'abord affecter une paire de variables pour chaque direction et son sens inverse. Ainsi, par exemple, la marche avant et la marche arrière forment une même paire.

Commodore 64

```
17 A(1)=5:B(1)=10:A(2)=10:B(2)=5
18 A(3)=6:B(3)=9:A(4)=9:B(4)=6
```

Les routines suivantes peuvent être ajoutées pour exécuter la séquence enregistrée de façon inverse.

```
92 GOSUB2000:REM EXECUTION INVERSE
2000 REM SP EXECUTION INVERSE
2010 FOR I=C TO 1 STEP -1
2020 FOR J=I TO 4
2030 IF DR(I,1)=A(J) THEN POKE DATREG,B(J):J=4
2040 NEXT J
2050 T=TI
2060 IF(TI-T) DR(I,2 THEN 2060
2070 NEXT I
2080 STOP
2090 RETURN
```

BBC Micro

```
1020  DIM DR(100,2),A(10),B(10)
1025 A(1)=5:B(1)=10:A(2)=10:B(2)=5
1026 A(B)=6:B(B)=9:A(4)=9:B(4)=6
1115 PROCexécution_inverse
2000 DEF PROCexécution_inverse
2010 FOR I=C TO 1 STEP-1
2020 FOR J=I TO 4
2030   IF DR(I,1)=A(B) THEN?DATREG=B(J):J=4
2040 NEXT J
2042 TIME=0
2045 REPEAT UNTIL TIME>= DR(I,2)
2047  ?DATREG=0
2050 NEXT I
2055 PRINT"TAPEZ C POUR CONTINUER"
2060 REPEAT A$=GET$
2070 UNTIL A$=*C*
2080 ENDPROC
3000 FORI=1TOC:PRINTDR(I,1),DR(I,2)
3010 NEXT
```

3) En supposant que la ligne 7 est réservée à la marche avant, la 6 à la marche arrière, la 5 à la gauche, et la 4 à la droite :

```
10 REM INTERRUPTEURS EXTERNES BBC
20 DDR=1FE62:DATREG=&FE60
30 ?DDR=15:REM LINES 4-7 INPUT
40 ?DATREG=0
50 PROCtest_entrée
60 GOTO50
70 :
80 DEF PROCtest_entrée
90 IF(?DATREG AND 240)=240 THEN ?DATREG=0
100 IF(?DATREG AND 128)=0THEN ?DATREG=5
110 IF(?DATREG AND 64)=0THEN ?DATREG=10
120 IF(?DATREG AND 32)=0THEN ?DATREG=6
130 IF(?DATREG AND 16)=0THEN ?DATREG=9
140 ENDPROC

10 REM INTERRUPTEURS EXTERNES CBM64
20 DDR=56579:DATREG=56577
30 POKEDDR,15:REM LINES 4-7 INPUT
40 POKEDATREG,0:REM MOTORS OFF
50 REM L7 FORWD,L6 REV,L5 LEFT, L4 RIGHT
55 REM UNE DES LIGNES EST-ELLE AU NIVEAU BAS?
60 IF(PEEK(DATREG)AND240)<>240 THENGOSUB1000:GOTO60
70 POKEDATREG,0:GOTO60
80 :
1000 REM SCAN INPUT LINES S/R
1005 IF PEEK(DATREG)AND128)=0THENPOKEDATRG,5
1010 IF PEEK(DATREG)AND64)=0THENPOKEDATREG,10
1020 IF PEEK(DATREG)AND32)=0THENPOKEDATREG,6
1030 IF PEEK(DATREG)AND16)=0THENPOKEDATREG,9
1040 RETURN
```

Connexions de saut

En poursuivant notre étude de l'adressage indexé sur le processeur 6809, nous considérons ici comment se servir de l'adressage indirect, et nous l'illustrons par l'écriture de caractères pour un affichage sur écran.

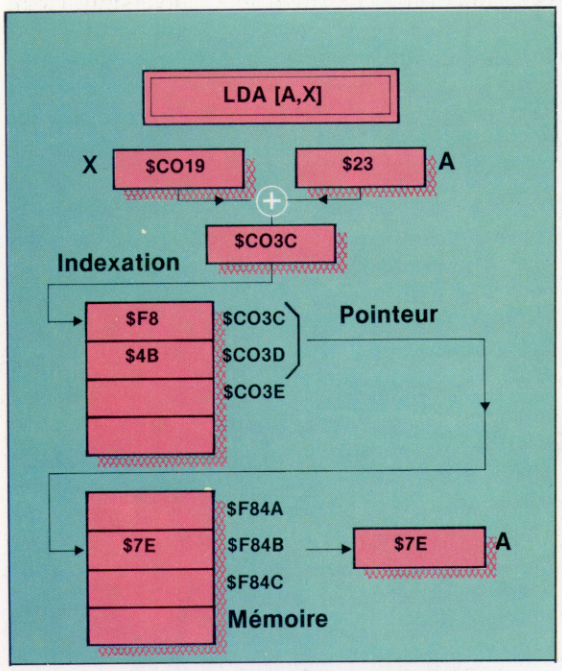
Tout d'abord, il faut préciser que l'adressage indirect n'est pas exactement un mode d'adressage en soi, mais qu'il s'agit d'un caractère supplémentaire pouvant être utilisé avec la plupart des autres modes ; c'est en fait un stade de calcul de l'adresse effective (l'adresse à partir de laquelle il faut chercher la donnée). L'adresse effective est calculée de l'une des manières que nous avons décrites (par accès direct, par adressage indexé ou par calcul effectif de l'adresse), mais si l'*indirection* est spécifiée, le contenu de l'adresse ainsi calculée et les emplacements mémoire suivants sont traités comme une adresse. C'est cette adresse qui devient l'adresse effective finale, à partir de laquelle la donnée est chargée.

Par exemple, si les valeurs suivantes sont stockées :

Adresse	Contenu
3000	40
3001	0A
400A	F2

l'instruction LDA $3000 chargera la valeur $40 dans l'accumulateur A, l'adresse effective étant $3000. L'indirection est toujours spécifiée en plaçant l'opérande entre crochets ; ainsi, LDA [$3000] chargera la valeur $F2 dans A, l'adresse effective étant la valeur stockée dans l'adresse qui est, à son tour, stockée en $3000 et $3001 — dans ce cas, $400A. Le contenu de $3000 et $3001 forme un pointeur (ou vecteur) de l'adresse effective, $400A. Notez que la convention du 6809 consiste à stocker les adresses avec l'octet hi avant l'octet lo : ainsi, $40 est stocké en $3000 et $0A en $3001. Telle est la convention hi-lo. Les processeurs Zilog Z80 et MOS Tech 6502 utilisent la convention opposée : $0A (octet lo de l'adresse) doit être stocké en $3000, et $40 (octet hi) en $3001. L'indirection peut souvent être utilisée en combinaison avec l'adressage indexé.

Le 6809 utilise moins l'adressage indirect que la plupart des processeurs (6502 et Z80, notamment), et ce, du fait de la richesse de ses modes d'adressage indexé. Il y a pourtant des situations dans lesquelles l'indirection peut être très utile ; l'une d'elles, dont nous parlerons plus tard, est l'utilisation de dispositifs d'interface périphériques. Les processeurs Motorola, contrairement aux familles 8080 et 8086 d'Intel, ont une *table d'implantation* des entrées/sorties (E/S) en mémoire. Les registres de communications dans les dispositifs d'interface apparaissent dans la table d'implantation principale du système, et les



Adressage indexé indirect
L'argument [A,X] de l'instruction LDA est entre crochets, ce qui signifie que le contenu de X ($CO19 ici) doit être additionné au contenu de A ($23) pour donner une adresse à 16 bits ($CO3C). Cet octet et le suivant ($CO3D) doivent être traités comme un pointeur de l'adresse de chargement effective ($F84B), dont le contenu ($7E) est finalement chargé en A. X étant additionné à A avant l'accès indirect, on appelle cela « indirection préindexée ». L'autre solution, la postindexation, nécessite que l'adresse indirecte soit calculée avant que l'indexation n'ait lieu.

valeurs peuvent y être stockées ou chargées, comme si c'était n'importe quel emplacement mémoire au lieu d'un canal vers le dispositif d'interface. Une routine pour contrôler l'un de ces dispositifs — par exemple, une routine d'affichage — nécessite l'adresse du registre d'interface du dispositif. Si celui-ci est translaté dans la table d'implantation ou s'il y a plus d'un dispositif de ce type, il est bien plus simple de changer l'un des emplacements mémoire qui contient l'adresse du registre communication du dispositif (un pointeur du dispositif) plutôt que d'avoir à trouver et à changer l'adresse de celui-ci chaque fois qu'on la rencontre. La routine renvoie indirectement au dispositif, en utilisant le pointeur.

L'utilisation la plus courante de cette technique apparaît dans une structure appelée *table de branchement,* qui n'est autre qu'une table de pointeurs. Tout système d'exploitation contient de nombreuses routines utiles qui exécutent les fonctions élémentaires de la machine — par exemple, lire un caractère du clavier ou afficher un caractère sur l'écran. La plupart des programmes en langage machine auront besoin d'utiliser ces routines à un moment ou à un autre. Le plus souvent, on y accède à l'aide d'une table de branchement, c'est-à-dire que les routines indiquées par les vecteurs de cette table peuvent être modifiées, ou translatées en mémoire, sans nécessiter

Accès indirect
Bien que l'adressage indirect soit essentiel pour les opérations d'ordinateur, il est difficile d'en trouver des exemples dans la vie de tous les jours. Voici toutefois une analogie : dans un service d'appel radio, si quelqu'un veut parler à un abonné, il ne l'appelle pas directement, mais il appelle le service d'appel, qui appelle à son tour l'abonné. C'est un système simple et souple, dans lequel le service d'appel fournit l'accès indirect (ou adressage) de ses abonnés.
(Cl. Pitchal/Rea, Rudman/Rea et Grégoire/Éd. Atlas.)

de changement dans les programmes qui les utilisent. Autrement dit, de telles routines sont toujours accessibles indirectement, par les pointeurs appropriés de la table de branchement. Lorsqu'une nouvelle version d'un système d'exploitation est conçue ou qu'une ROM est modernisée, il est rare pour ces routines de S.E. de rester dans leur position d'origine; mais si la table de branchement demeure et que ses pointeurs sont modifiés de façon à refléter les nouvelles adresses de routines S.E., tout logiciel écrit pour l'ancien S.E. demeurera valable sur le nouveau système.

Dans de nombreux S.E., il est fréquent d'avoir un point d'entrée et de faire tous les appels de sous-programmes à cette adresse. L'un des registres d'UC est en outre utilisé pour passer un code de fonction qui sert à déterminer quelle routine sera appelée. Le code de fonction est utilisé comme un index ou décalé dans le vecteur approprié de la table de branchement, et le contrôle est transféré par ce pointeur à la routine désirée.

A titre d'exemple, supposons que nous ayons 4 K de ROM, localisés en $F000, dont les 256 premiers octets ($F000 à $F0FF) contiennent une table comportant jusqu'à 128 adresses de sous-programmes stockés en ROM. La routine d'entrée (l'adresse par laquelle toutes les routines S.E. sont adressées) est localisée en $F100, et elle attend qu'un code de fonction compris entre 0 et 127 soit stocké dans l'accumulateur B; ce code sert à la routine d'entrée à passer le contrôle aux sous-programmes appropriés, puis à le repasser au programme appelant, une fois que l'exécution est achevée. La routine d'appel pour le numéro de fonction 1 est :

LDB #1	Met le code de fonction en B.
JSR $F100	Appelle le sous-programme d'entrée.

La routine d'entrée est la suivante :

LDX $F000	Adresse de début de la table de branchement.
LSLB	Décale B d'une case vers la gauche (ce qui équivaut à multiplier le contenu de B par 2), puisque chaque entrée de la table a 2 bits de longueur. Ainsi, le pointeur approprié au code de fonction 1 est stocké en $F002 et $F003, tandis que le pointeur pour le code 2 est en $F004 et $F005, etc.
BRA [B,X]	Transfère le contrôle à l'adresse trouvée à la Bième position dans la table.

Notre exemple suivant montre un autre usage possible de l'adressage indirect avec un écran d'affichage de table d'implantation; sur beaucoup de micros, la mémoire écran occupe un bloc de la mémoire centrale, et on peut y accéder directement pour plus de rapidité. Pour simplifier, supposons que l'écran occupe un bloc de mémoire compris entre $E000 et $E3FF, représentant 16 lignes de 64 caractères. La position du curseur est une valeur à 16 bits dans cet intervalle, localisée en $E400. Le premier sous-programme efface l'écran en écrivant un caractère espace (code ASCII 32) en chaque position de caractère. Le second sous-programme écrit le caractère passé dans A à l'écran dans la position actuelle du curseur, à moins que ce caractère soit un retour chariot (ASCII 13), auquel cas il effacera le reste de la ligne et positionnera le curseur au début de la ligne suivante. Le curseur est représenté par le tiret de soulignage («–») dans cet exemple.

SPACE	EQU 32	Code ASCII pour espace.
CR	EQU 13	Code ASCII pour retour chariot.
HOME	EQU $E000	Début de mémoire écran.
LENGTH	EQU 1024	Taille de mémoire écran (16 lignes × 64 caractères = 1024).
CURSOR	EQU $E400	$E400 et $E401 pointent l'adresse actuelle du curseur en zone mémoire écran.
	ORG $1000	
CURCHR	FCB 95	Tiret de soulignage (ASCII 95).

*****Sous-programme pour effacer l'écran*****

LDA #SPACE	Caractère espace en A.
LDX #HOME	Curseur en début d'écran.
STX CURSOR	Stocke position actuelle de curseur en CURSOR (c'est-à-dire $E400, $E401).

```
        LDB #LENGTH      Taille d'écran en B.
LOOP1   STA [CURSOR]     Stocke un espace dans la
                         position actuelle du
                         curseur.
        INC CURSOR       Incrémente position
                         curseur.
        DECB             Décrémente de la quantité
                         de mémoire écran restant
                         entre position curseur et
                         fin de mémoire écran.
        BGT LOOP1        Prochain espace, jusqu'à
                         ce qu'il ne reste plus de
                         mémoire écran.
        STX CURSOR       Retour curseur en début
                         d'écran.
        LDA CURSOR       Code ASCII de curseur
                         en A.
        STA [CURSOR]     Stocke caractère curseur
                         en position actuelle de
                         curseur.
        RTS
```

*****Sous-programme pour afficher caractère en A, si affichable*****

```
        CMPA SPACE       Espace est le premier
                         caractère imprimable en
                         ASCII.
        BLT NOTP         Si accumulateur contient
                         valeur ASCII inférieure à 32,
                         ce n'est pas imprimable,
                         donc aller en NOTP.
        STA [CURSOR]     Stocke en position
                         actuelle de curseur.
        INC CURSOR       Incrémente position
                         curseur.
CHKEOS  LDX #HOME        Vérifier si fin d'écran.
        LEAY #LENGTH, X  Fin d'écran en Y.
        CMPY CURSOR      Si position curseur
                         supérieure à fin d'écran,
                         alors...
        BGT FINISH       On a atteint la fin de
                         l'écran, donc aller en FINISH.
```

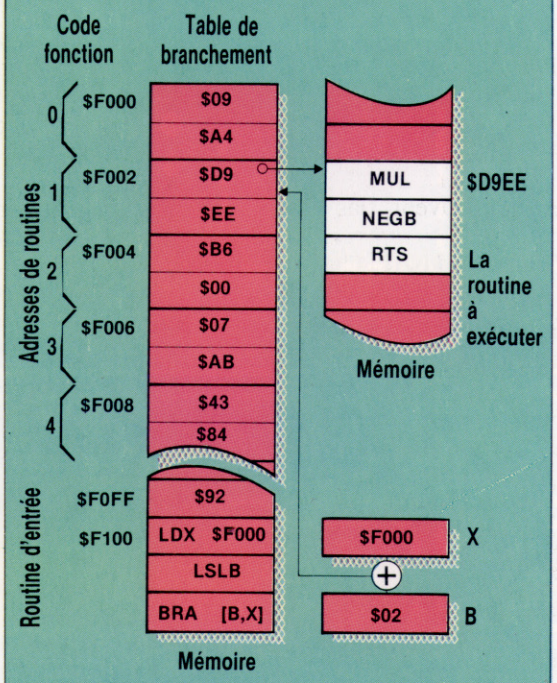
*****Sous-programme pour faire défiler l'écran*****

```
SCROLL  LEAY 64, X       Y est à une longueur d'une
                         ligne de X (fin de mémoire
                         écran).
        LDB #LENGTH      Calcule la quantité à faire
                         défiler.
        SUBB #64         Soustrait 64 de la
                         longueur.
LOOP2   LDA, Y+          Recule caractères d'une
                         ligne (notez
                         l'auto-incrémentation;
                         v. page 618).
        STA,X+
        DECB
        BGT LOOP2        Boucle jusqu'à défilement
                         complet.
        LDD CURSOR       Curseur au début de la
                         dernière ligne.
        SUBD #64
        STD CURSOR
        BRA FINISH
```

*****Sous-programme pour vérifier le retour chariot*****

```
NOTP    CMPA #CR         Ce caractère non
                         imprimable est-il
                         un retour chariot?
        BNE FINISH       Ignoré sinon.
        LDD CURSOR       Vous pouvez trouver
        ANDB #%11100000  comment ceci donne le
                         début de la ligne suivante
                         (notez le masque binaire
                         AND).
        ADDD #64
        STD CURSOR
        BRA CHKEOS       Vérifie si fin d'écran.
FINISH  LDA CURSOR       Caractère curseur en A.
        STA [CURSOR]     Stocke en position
                         actuelle de curseur.
        RTS
```

Table de branchement



La table de branchement est ici une liste de 128 pointeurs d'adresses à 2 octets, situés entre $F000 et $F0FF. Chacun de ces pointeurs contient l'adresse de départ d'une routine quelque part en mémoire. Pour exécuter l'une de ces routines, il nous faut seulement charger l'accumulateur B avec un code fonction ($01, par exemple) qui identifie la routine désirée (localisée en $D9EE dans ce cas) et puis JSR (saute) à la routine d'entrée, dont l'adresse de début est ici $F0FF. Nous supposons que ces routines sont en ROM (car elles font partie du logiciel basé en ROM, ainsi que le système d'exploitation); on pourra donc chercher le code fonction et l'adresse de départ de la routine d'entrée dans le manuel du programmeur.
La routine d'entrée multiplie le code fonction par 2 et l'utilise comme décalé pour l'adresse de début de table, afin de trouver le pointeur d'adresse de routine désiré : le pointeur de la routine $01 est en $F002, par exemple (=$F000+2×$01), le pointeur de la routine $02 est en $F004 (=$F000+2×$02), et ainsi de suite. Le pointeur est ensuite utilisé par la routine d'entrée dans une instruction de branchement indirect, afin de passer le contrôle à la routine en cours en $D9EE. Notez que la routine d'entrée se branche sur (plutôt qu'appelle) la routine d'exécution, de sorte que, lorsqu'on rencontre RTS, le contrôle revient au point du programme d'où la routine d'entrée fut appelée la première fois.
Une table de branchement présente l'avantage de permettre au programmeur de réétudier et de relocaliser les routines qu'il adresse, tout en permettant aux programmes écrits auparavant de tourner sur le nouveau système : les codes fonction et l'adresse de la routine d'entrée sont maintenus constants durant toute la vie du système, mais le contenu des pointeurs d'adresse de la table de branchement et l'emplacement de la table elle-même peuvent changer à volonté.

L'origine des espèces

Jeu d'arcades extrêmement célèbre, Pacman d'Atarisoft arrive en version « officielle », pour le Spectrum et le Vic-20, afin de lutter contre les innombrables copies.

Pacman est l'ancêtre des jeux de labyrinthe; de nombreuses copies existent sous des formes diverses. A chaque fois, le personnage principal doit traverser un réseau de chambres ou de couloirs, amasser des trésors et éviter de redoutables adversaires. Il lui est même parfois possible de les attaquer : dans Sabre Wulf, s'emparer de certaines orchidées permet au joueur d'être invisible pour les créatures de la jungle. C'est là une idée qui vient des jeux d'aventures, dans lesquels la possession d'une épée magique (ou d'un objet analogue) est toujours d'un grand secours.

Pacman est une sorte de camembert de couleur jaune, installé au départ au centre de l'écran; le joueur doit le guider dans le labyrinthe en avalant les pilules qui jonchent son chemin. Il ne faut jamais perdre de vue les fantômes lancés à la poursuite du héros. Certaines pilules sont magiques, et, si Pacman en avale une, il peut, pour peu de temps, pourchasser et dévorer ses adversaires; il doit aussi s'efforcer de gober les fruits qui, parfois, font leur apparition ça et là; cela lui vaudra des points.

Les copies d'Atari

Il peut paraître absurde de comparer entre elles des versions différentes d'un même jeu, car leur réalisation est étroitement liée aux caractéristiques de l'appareil auquel elles sont destinées. Cela est particulièrement vrai pour les ordinateurs Atari, qui ont la réputation, largement justifiée, d'être d'excellente qualité (mais qui sont peut-être un peu chers).

Cela dit, il faut bien admettre que les deux variantes de Pacman qu'on nous propose ne sont pas à la hauteur de l'original.

Sur les ordinateurs Atari, le labyrinthe est vaste, le son et le graphisme excellents, le déplacement des lutins se fait en douceur. On ne peut en dire autant des versions Spectrum et Vic-20. En ce qui concerne ce dernier, il était évidemment très difficile de surmonter les problèmes liés à l'exiguïté de sa mémoire : aussi le labyrinthe est-il ridiculement petit, tandis que les fantômes sont tellement grands qu'il est très difficile de les éviter. Malgré ce genre d'inconvénient « surprenant », le graphisme et le son sont réussis.

Si les instructions sur le Spectrum sont plus complètes et s'il existe une option clavier (ce qui n'est pas le cas pour les autres), le son est rudimentaire. Il est vrai que le meilleur programmeur du monde ne pourra jamais faire des miracles avec des possibilités sonores aussi réduites que celles de l'appareil. Quant au graphisme, il est particulièrement décevant : le labyrinthe est bien celui de l'original, mais un scintillement incessant fait irrésistiblement penser à un film muet. Si l'on comprend parfaitement que la version Vic-20 souffre des limites mêmes de l'engin, on ne s'explique pas la médiocrité du jeu pour le Spectrum.

Lors de son apparition, en 1980, Pacman connut un succès phénoménal, mais on a fait beaucoup mieux depuis. Atarisoft prit soin de menacer (en pure perte, d'ailleurs) de procès les autres compagnies qui sortiraient des programmes du même genre; quoi qu'il en soit, la firme a beaucoup trop tardé à mettre en vente les versions « officielles » de ce jeu, alors qu'elle aurait pu en inonder le marché. Il existe aujourd'hui bien des jeux d'arcades infiniment plus sophistiqués. Que les possesseurs de Spectrum et de Vic-20 ne se précipitent pas sur ce jeu !

Pacman : ordinateurs Atari, ZX Spectrum, Vic-20.
Éditeur : Atari Corporation.
Auteur : Atari.
Manche à balai : indispensable.
Format : cartouche (Atari et Vic-20) ou cassette (Spectrum).

Liz Dixon

Page manquante
(publicité)

Page manquante
(publicité)